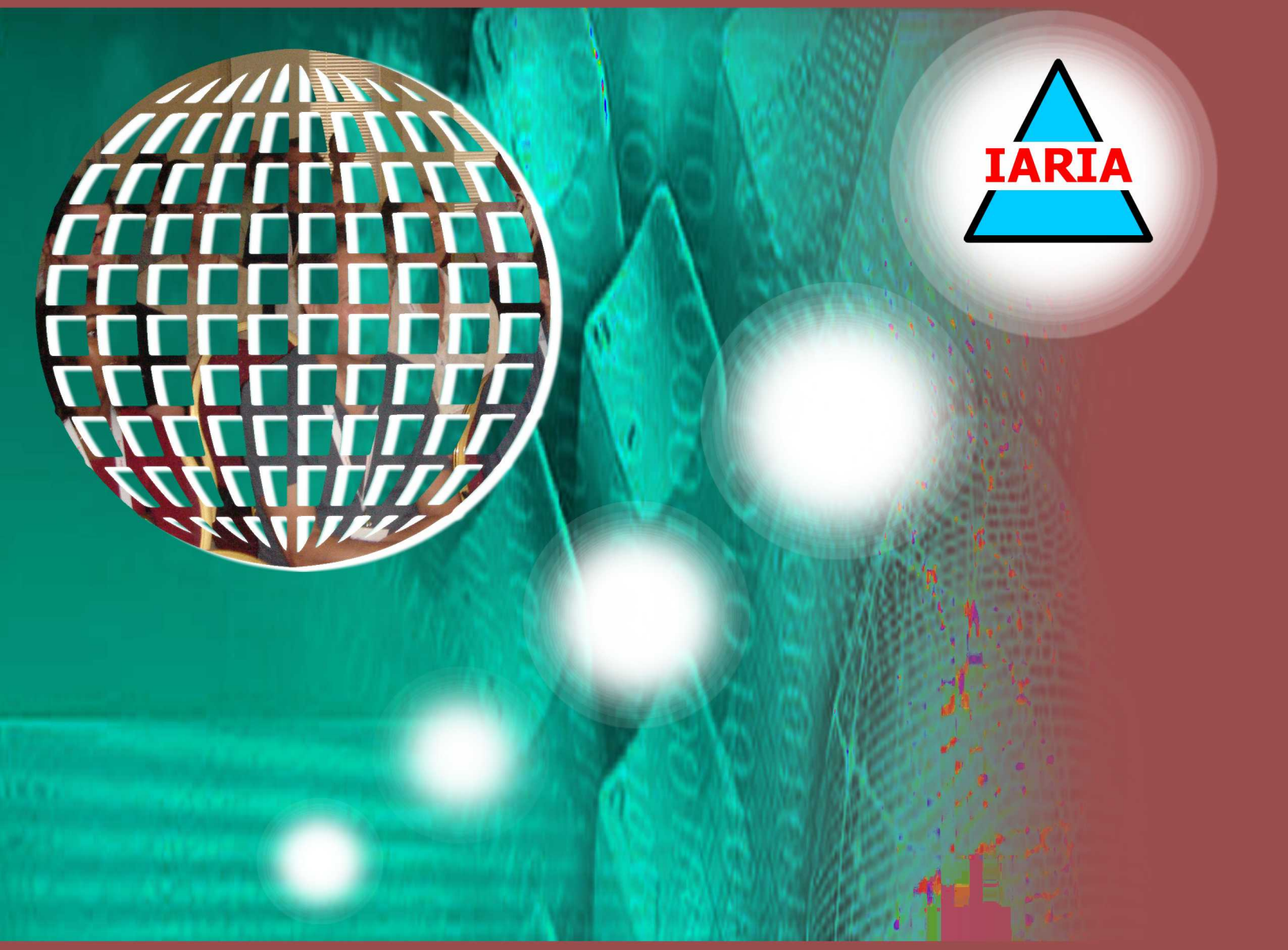
International Journal on Advances in Software



2015 vol. 8 nr. 1&2

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 8, Number 1 & 2, 2015

Hongyu Pei Breivold, ABB Corporate Research, Sweden
Carsten Brockmann, Universität Potsdam, Germany
Antonio Bucchiarone, Fondazione Bruno Kessler, Italy
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria
Dumitru Burdescu, University of Craiova, Romania
Martine Cadot, University of Nancy / LORIA, France
Isabel Candal-Vicente, Universidad del Este, Puerto Rico
Juan-Vicente Capella-Hernández, Universitat Politècnica de València, Spain
Jose Carlos Metrolho, Polytechnic Institute of Castelo Branco, Portugal
Alain Casali, Aix-Marseille University, France
Yaser Chaaban, Leibniz University of Hanover, Germany
Patryk Chamuczyński, Radytek, Poland
Savvas A. Chatzichristofis, Democritus University of Thrace, Greece
Antonin Chazalet, Orange, France
Jiann-Liang Chen, National Dong Hwa University, China
Shiping Chen, CSIRO ICT Centre, Australia
Wen-Shiung Chen, National Chi Nan University, Taiwan
Zhe Chen, College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China PR
Po-Hsun Cheng, National Kaohsiung Normal University, Taiwan
Yoonsik Cheon, The University of Texas at El Paso, USA
Lau Cheuk Lung, INE/UFSC, Brazil
Robert Chew, Lien Centre for Social Innovation, Singapore
Andrew Connor, Auckland University of Technology, New Zealand
Rebeca Cortázar, University of Deusto, Spain
Noël Crespi, Institut Telecom, Telecom SudParis, France
Carlos E. Cuesta, Rey Juan Carlos University, Spain
Duilio Curcio, University of Calabria, Italy
Mirela Danubianu, "Stefan cel Mare" University of Suceava, Romania
Paulo Asterio de Castro Guerra, Tapijara Programação de Sistemas Ltda. - Lambari, Brazil
Cláudio de Souza Baptista, University of Campina Grande, Brazil
Maria del Pilar Angeles, Universidad Nacional Autonónoma de México, México
Rafael del Vado Vírseda, Universidad Complutense de Madrid, Spain
Giovanni Denaro, University of Milano-Bicocca, Italy
Hepu Deng, RMIT University, Australia
Nirmit Desai, IBM Research, India
Vincenzo Deufemia, Università di Salerno, Italy
Leandro Dias da Silva, Universidade Federal de Alagoas, Brazil
Javier Diaz, Rutgers University, USA
Nicholas John Dingle, University of Manchester, UK
Roland Dodd, CQUniversity, Australia
Aijuan Dong, Hood College, USA
Suzana Dragicevic, Simon Fraser University- Burnaby, Canada
Cédric du Mouza, CNAM, France
Ann Dunkin, Palo Alto Unified School District, USA
Jana Dvorakova, Comenius University, Slovakia

Lars Ebrecht, German Aerospace Center (DLR), Germany
Hans-Dieter Ehrich, Technische Universität Braunschweig, Germany
Jorge Ejarque, Barcelona Supercomputing Center, Spain
Atilla Elçi, Aksaray University, Turkey
Khaled El-Fakih, American University of Sharjah, UAE
Gledson Elias, Federal University of Paraíba, Brazil
Sameh Elnikety, Microsoft Research, USA
Fausto Fasano, University of Molise, Italy
Michael Felderer, University of Innsbruck, Austria
João M. Fernandes, Universidade de Minho, Portugal
Luis Fernandez-Sanz, University of de Alcala, Spain
Felipe Ferraz, C.E.S.A.R, Brazil
Adina Magda Florea, University "Politehnica" of Bucharest, Romania
Wolfgang Fohl, Hamburg Universiy, Germany
Simon Fong, University of Macau, Macau SAR
Gianluca Franchino, Scuola Superiore Sant'Anna, Pisa, Italy
Naoki Fukuta, Shizuoka University, Japan
Martin Gaedke, Chemnitz University of Technology, Germany
Félix J. García Clemente, University of Murcia, Spain
José García-Fanjul, University of Oviedo, Spain
Felipe Garcia-Sanchez, Universidad Politecnica de Cartagena (UPCT), Spain
Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany
Tejas R. Gandhi, Virtua Health-Marlton, USA
Andrea Giachetti, Università degli Studi di Verona, Italy
Robert L. Glass, Griffith University, Australia
Afzal Godil, National Institute of Standards and Technology, USA
Luis Gomes, Universidade Nova Lisboa, Portugal
Diego Gonzalez Aguilera, University of Salamanca - Avila, Spain
Pascual Gonzalez, University of Castilla-La Mancha, Spain
Björn Gottfried, University of Bremen, Germany
Victor Govindaswamy, Texas A&M University, USA
Gregor Grambow, University of Ulm, Germany
Carlos Granell, European Commission / Joint Research Centre, Italy
Christoph Grimm, University of Kaiserslautern, Austria
Michael Grottke, University of Erlangen-Nuernberg, Germany
Vic Grout, Glyndwr University, UK
Ensar Gul, Marmara University, Turkey
Richard Gunstone, Bournemouth University, UK
Zhensheng Guo, Siemens AG, Germany
Phuong H. Ha, University of Tromso, Norway
Ismail Hababeh, German Jordanian University, Jordan
Shahliza Abd Halim, Lecturer in Universiti Teknologi Malaysia, Malaysia
Herman Hartmann, University of Groningen, The Netherlands
Jameleddine Hassine, King Fahd University of Petroleum & Mineral (KFUPM), Saudi Arabia
Tzung-Pei Hong, National University of Kaohsiung, Taiwan
Peizhao Hu, NICTA, Australia

Chih-Cheng Hung, Southern Polytechnic State University, USA
Edward Hung, Hong Kong Polytechnic University, Hong Kong
Noraini Ibrahim, Universiti Teknologi Malaysia, Malaysia
Anca Daniela Ionita, University "POLITEHNICA" of Bucharest, Romania
Chris Ireland, Open University, UK
Kyoko Iwasawa, Takushoku University - Tokyo, Japan
Mehrshid Javanbakht, Azad University - Tehran, Iran
Wassim Jaziri, ISIM Sfax, Tunisia
Dayang Norhayati Abang Jawawi, Universiti Teknologi Malaysia (UTM), Malaysia
Jinyuan Jia, Tongji University. Shanghai, China
Maria Joao Ferreira, Universidade Portucalense, Portugal
Ahmed Kamel, Concordia College, Moorhead, Minnesota, USA
Teemu Kanstrén, VTT Technical Research Centre of Finland, Finland
Nittaya Kerdprasop, Suranaree University of Technology, Thailand
Ayad ali Keshlaf, Newcastle University, UK
Nhien An Le Khac, University College Dublin, Ireland
Sadegh Kharazmi, RMIT University - Melbourne, Australia
Kyoung-Sook Kim, National Institute of Information and Communications Technology, Japan
Youngjae Kim, Oak Ridge National Laboratory, USA
Roger "Buzz" King, University of Colorado at Boulder, USA
Cornel Klein, Siemens AG, Germany
Alexander Knapp, University of Augsburg, Germany
Radek Koci, Brno University of Technology, Czech Republic
Christian Kop, University of Klagenfurt, Austria
Michal Krátký, VŠB - Technical University of Ostrava, Czech Republic
Narayanan Kulathuramaiyer, Universiti Malaysia Sarawak, Malaysia
Satoshi Kurihara, Osaka University, Japan
Eugenijus Kurilovas, Vilnius University, Lithuania
Philippe Lahire, Université de Nice Sophia-Antipolis, France
Alla Lake, Linfo Systems, LLC, USA
Fritz Laux, Reutlingen University, Germany
Luigi Lavazza, Università dell'Insubria, Italy
Fábio Luiz Leite Júnior, Universidade Estadual da Paraiba,Brazil
Alain Lelu, University of Franche-Comté / LORIA, France
Cynthia Y. Lester, Georgia Perimeter College, USA
Clement Leung, Hong Kong Baptist University, Hong Kong
Weidong Li, University of Connecticut, USA
Corrado Loglisci, University of Bari, Italy
Francesco Longo, University of Calabria, Italy
Sérgio F. Lopes, University of Minho, Portugal
Pericles Loucopoulos, Loughborough University, UK
Alen Lovrencic, University of Zagreb, Croatia
Qifeng Lu, MacroSys, LLC, USA
Xun Luo, Qualcomm Inc., USA
Shuai Ma, Beihang University, China
Stephane Maag, Telecom SudParis, France

Ricardo J. Machado, University of Minho, Portugal
Maryam Tayefeh Mahmoudi, Research Institute for ICT, Iran
Nicos Malevris, Athens University of Economics and Business, Greece
Herwig Mannaert, University of Antwerp, Belgium
José Manuel Molina López, Universidad Carlos III de Madrid, Spain
Francesco Marcelloni, University of Pisa, Italy
Eda Marchetti, Consiglio Nazionale delle Ricerche (CNR), Italy
Leonardo Mariani, University of Milano Bicocca, Italy
Gerasimos Marketos, University of Piraeus, Greece
Abel Marrero, Bombardier Transportation, Germany
Adriana Martin, Universidad Nacional de la Patagonia Austral / Universidad Nacional del Comahue, Argentina
Goran Martinovic, J.J. Strossmayer University of Osijek, Croatia
Paulo Martins, University of Trás-os-Montes e Alto Douro (UTAD), Portugal
Stephan Mäs, Technical University of Dresden, Germany
Constandinos Mavromoustakis, University of Nicosia, Cyprus
Jose Merseguer, Universidad de Zaragoza, Spain
Seyedeh Leili Mirtaheri, Iran University of Science & Technology, Iran
Lars Moench, University of Hagen, Germany
Yasuhiko Morimoto, Hiroshima University, Japan
Antonio Navarro Martín, Universidad Complutense de Madrid, Spain
Filippo Neri, University of Naples, Italy
Muaz A. Niazi, Bahria University, Islamabad, Pakistan
Natalja Nikitina, KTH Royal Institute of Technology, Sweden
Roy Oberhauser, Aalen University, Germany
Pablo Oliveira Antonino, Fraunhofer IESE, Germany
Rocco Oliveto, University of Molise, Italy
Sascha Opletal, Universität Stuttgart, Germany
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Claus Pahl, Dublin City University, Ireland
Marcos Palacios, University of Oviedo, Spain
Constantin Paleologu, University Politehnica of Bucharest, Romania
Kai Pan, UNC Charlotte, USA
Yiannis Papadopoulos, University of Hull, UK
Andreas Papasalouros, University of the Aegean, Greece
Rodrigo Paredes, Universidad de Talca, Chile
Päivi Parviainen, VTT Technical Research Centre, Finland
João Pascoal Faria, Faculty of Engineering of University of Porto / INESC TEC, Portugal
Fabrizio Pastore, University of Milano - Bicocca, Italy
Kunal Patel, Ingenuity Systems, USA
Óscar Pereira, Instituto de Telecomunicacoes - University of Aveiro, Portugal
Willy Picard, Poznań University of Economics, Poland
Jose R. Pires Manso, University of Beira Interior, Portugal
Sören Pirk, Universität Konstanz, Germany
Meikel Poess, Oracle Corporation, USA
Thomas E. Potok, Oak Ridge National Laboratory, USA
Christian Prehofer, Fraunhofer-Einrichtung für Systeme der Kommunikationstechnik ESK, Germany

Ela Pustułka-Hunt, Bundesamt für Statistik, Neuchâtel, Switzerland
Mengyu Qiao, South Dakota School of Mines and Technology, USA
Kornelije Rabuzin, University of Zagreb, Croatia
J. Javier Rainer Granados, Universidad Politécnica de Madrid, Spain
Muthu Ramachandran, Leeds Metropolitan University, UK
Thurasamy Ramayah, Universiti Sains Malaysia, Malaysia
Prakash Ranganathan, University of North Dakota, USA
José Raúl Romero, University of Córdoba, Spain
Henrique Rebêlo, Federal University of Pernambuco, Brazil
Hassan Reza, UND Aerospace, USA
Elvinia Riccobene, Università degli Studi di Milano, Italy
Daniel Riesco, Universidad Nacional de San Luis, Argentina
Mathieu Roche, LIRMM / CNRS / Univ. Montpellier 2, France
José Rouillard, University of Lille, France
Siegfried Rouvrais, TELECOM Bretagne, France
Claus-Peter Rückemann, Leibniz Universität Hannover / Westfälische Wilhelms-Universität Münster / North-German Supercomputing Alliance, Germany
Djamel Sadok, Universidade Federal de Pernambuco, Brazil
Ismael Sanz, Universitat Jaume I, Spain
M. Saravanan, Ericsson India Pvt. Ltd -Tamil Nadu, India
Idrissa Sarr, University of Cheikh Anta Diop, Dakar, Senegal / University of Quebec, Canada
Patrizia Scandurra, University of Bergamo, Italy
Giuseppe Scanniello, Università degli Studi della Basilicata, Italy
Daniel Schall, Vienna University of Technology, Austria
Rainer Schmidt, Munich University of Applied Sciences, Germany
Cristina Seceleanu, Mälardalen University, Sweden
Sebastian Senge, TU Dortmund, Germany
Isabel Seruca, Universidade Portucalense - Porto, Portugal
Kewei Sha, Oklahoma City University, USA
Simeon Simoff, University of Western Sydney, Australia
Jacques Simonin, Institut Telecom / Telecom Bretagne, France
Cosmin Stoica Spahiu, University of Craiova, Romania
George Spanoudakis, City University London, UK
Alin Stefanescu, University of Pitesti, Romania
Lena Strömbäck, SMHI, Sweden
Osamu Takaki, Japan Advanced Institute of Science and Technology, Japan
Antonio J. Tallón-Ballesteros, University of Seville, Spain
Wasif Tanveer, University of Engineering & Technology - Lahore, Pakistan
Ergin Tari, Istanbul Technical University, Turkey
Steffen Thiel, Furtwangen University of Applied Sciences, Germany
Jean-Claude Thill, Univ. of North Carolina at Charlotte, USA
Pierre Tiako, Langston University, USA
Božo Tomas, HT Mostar, Bosnia and Herzegovina
Davide Tosi, Università degli Studi dell'Insubria, Italy
Guglielmo Trentin, National Research Council, Italy
Dragos Truscan, Åbo Akademi University, Finland

Chrisa Tsinaraki, Technical University of Crete, Greece
Roland Ukor, FirstLinq Limited, UK
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria
José Valente de Oliveira, Universidade do Algarve, Portugal
Dieter Van Nuffel, University of Antwerp, Belgium
Shirshu Varma, Indian Institute of Information Technology, Allahabad, India
Konstantina Vassilopoulou, Harokopio University of Athens, Greece
Miroslav Velev, Aries Design Automation, USA
Tanja E. J. Vos, Universidad Politécnica de Valencia, Spain
Krzysztof Walczak, Poznan University of Economics, Poland
Jianwu Wang, San Diego Supercomputer Center / University of California, San Diego, USA
Yandong Wang, Wuhan University, China
Rainer Weinreich, Johannes Kepler University Linz, Austria
Stefan Wesarg, Fraunhofer IGD, Germany
Wojciech Wiza, Poznan University of Economics, Poland
Martin Wojtczyk, Technische Universität München, Germany
Hao Wu, School of Information Science and Engineering, Yunnan University, China
Mudasser F. Wyne, National University, USA
Zhengchuan Xu, Fudan University, P.R.China
Yiping Yao, National University of Defense Technology, Changsha, Hunan, China
Stoyan Yordanov Garbatov, Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-ID, Portugal
Weihai Yu, University of Tromsø, Norway
Wenbing Zhao, Cleveland State University, USA
Hong Zhu, Oxford Brookes University, UK
Qiang Zhu, The University of Michigan - Dearborn, USA

International Journal on Advances in Software
Volume 8, Numbers 1 & 2, 2015

CONTENTS

Creating and Using Personas in Software Development Practice

Jane Billestrup, Jan Stage
Department of Computer Science
Aalborg University
Aalborg, Denmark
{jane,jans}@cs.aau.dk

Lene Nielsen, Kira Storgaard Hansen
Games & Interaction Design
IT University of Copenhagen
Copenhagen, Denmark
{lene,kist}@itu.dk

Abstract — **Personas has been suggested as a strong technique for providing software developers with a deep understanding of the prospective users of a software system. This paper reports from two separate but related empirical studies. The first study was a questionnaire survey about Personas usage in software development companies. The purpose was to uncover to what extent and in which ways Personas are used in software development companies located in a specific geographical area. This study demonstrated that less than half of the respondents had ever heard about Personas. We also identified key obstacles towards use of the technique: lack of knowledge of the technique, lack of resources, sparse descriptions and scarce integration in development. The second study was based on detailed interviews with four software developers about their usage of Personas in development processes in the software industry. We identified basic practices in Personas creation and usage, and found that the respondents understand Personas creation and use differently from the practice described in the literature. In fact, developers are evolving their own practices for creating and using Personas.**

Keywords—Personas; Personas creation and use; software development; questionnaire survey; interview.

I. INTRODUCTION

This paper is an extended version of the paper "Creating and Using Personas in Software Development Practice: Advantages, Obstacles and Experiences" [1].

Personas is being promoted as a technique that supports design and engineering of interactive software systems with an explicit focus on the prospective end-users.

The general definition of the technique is that a Persona is a description of a fictitious person based on data collected about the target user group of a system [2][3]. The common way to represent a Persona is as a text describing, and usually also a photo depicting, the fictitious person [2][4].

The main idea for introducing Personas is consistent with results from numerous reports that have documented that software developers lack knowledge and understanding of their users, their work, and their goals, e.g., [5][6]. A consequence is that when a system has been developed, it does not fulfil the needs of the users and is incompatible with their work processes. The Personas technique has been suggested as a strong tool to overcome these problems by providing software developers with a specific understanding of prospective end-users of their software [7].

It has been argued that the use of Personas provides software developers with empathy for, and engagement in, the end-users of the software solution [8]. There are also literature that concludes that the use of Personas has been a success [9][10].

The literature includes several conclusions about the benefits of the Personas technique, if it is used to its full potential. Matthews et al. [11] found that the designers who had a very positive attitude towards Personas were primarily those who had done extensive work with Personas, and had some training in the creation of Personas, and used them as prescribed by the literature. The Personas technique is not yet incorporated as an integrated and general part of the toolbox in the software development industry [11]. It has been documented that a main reason for this is that many developers in the industry have problems using Personas in practice [12]. Thus, there are still many unanswered questions about the actual advantages of using Personas in software development practice. The strength of using Personas compared to other techniques are also unexplored.

The purpose of this paper is to inquire into the way in which software companies use Personas and whether the technique is used as proposed in the literature. We report from a questionnaire survey and a case study of experiences with creation and use of Personas in software development practice. The questionnaire survey (Study A) was conducted in a delimited region in Denmark, where we inquired into the experiences software companies in this region had in using Personas and incorporating the technique as a part of their development toolbox. The case study (Study B) was based on interviews with four developers who were or had been working with Personas in practice. Our focus in this paper is on comparing the literature with the experiences and the perceived strengths and weaknesses of the Personas technique from the perspective of the software development industry, Our empirical basis includes using a mixed method approach involving both quantitative and qualitative data collection.

Section II presents a more detailed description of work related to this study. It describes how Personas are created and used, including the pitfalls to avoid. Section III describes the method used in the questionnaire survey (Study A). Section IV presents the results from this survey. Section V presents the method used in the case study (Study B). Section VI provides the findings derived from the interviews. Section VII compares the findings from the two studies and discusses the results compared to experiences about Personas

reported in the literature. Finally, Section VIII provides the conclusion.

II. Related Work

The literature offers four different perspectives regarding the basis for and role of Personas [13]: 1) Cooper's goal-directed perspective 2) Grudin, Pruitt and Adlin's role-based perspective 3) the engaging perspective, which emphasises how the story can engage the reader. These three perspectives agree that the Persona descriptions should be founded on real data. However, 4) the fiction-based perspective, does not include data as a basis for Persona description, but creates Personas from the designers' intuition and assumptions. Even though the Personas technique has been around for more than a decade, when comparing the four perspectives, it is still unclear what and how much background material is required to create Personas [14].

The common perceived benefits of Personas, when designing products are two-fold: 1) the technique facilitates that designers remember that they are different from the end-users, and 2) the technique enables designers to envision the end-users' needs and wants. Furthermore, in the design process Personas increase the focus on users' and their needs. The technique is an effective communication tool, which uses the Persona description to acquire direct design influence and lead to better design decisions and definition of the products' feature set [2][3][7][10][15][16][17].

The literature includes a rich variety of guidelines and experiences about the use of Personas.

1) Defining Personas. The literature originally defined a Persona as a text and a photo describing the character [2] [18]. Later developed into posters, websites and hand-outs [19]. Personas are considered to be most useful if they are developed as whole characters, described with enough detail for designers and developers to get a feeling of its personality [7][12][19]. The benefits of Personas are that they enable designers to envision the end-user's needs and wants, reminding designers that their own needs are not necessarily the end-users' needs, and provide an effective communication tool, which facilitates better design decisions [10][15] [16][17].

2) Creating Personas. Before creating Personas, a comprehensive study of the target user group is suggested. It has been recommended to acquire this information through interviews with the target user group [20] or observational studies of them [21]. Yet Chapman and Milham argue that it is not possible to verify that the created Personas actually reflect the target user group [22]. It has been suggested to create 3-5 Personas [23][24], but the amount of users one Persona can represent has been questioned [22].

3) Personas Critique. Personas has been characterized as unreliable and preventing designers from meeting actual users [5][12][13]. Problems have been reported regarding creation and distribution of the developed Personas [12] [19]. The descriptions have been perceived as unreliable and not well communicated. Also, developers lack understanding of how to use Personas [3][12][19]. The technique itself is criticised for being too founded on qualitative data and, as a consequence of that, being non-scientific, being difficult to implement. Also, for not being able to describe actual people as it only portrays some characteristics, and for preventing designers from meeting actual users [5]. Moreover, the unsolved question about how many users one Persona can represent is emphasized as problematic [22].

Some have tried to prevent poor use of the Personas technique, e.g., Faily and Flechais [25] describe regularly sending information about the Personas to the development team, to ensure that the designers and developers consider the Personas in the design process. They also suggest that the creators should hand over instructions and provide tools that support the developers' usage [25]. Problems in applying Personas are reported as also involving the mindset of the developers, which is documented by both Blomquist and Arvola [6], and Pruitt and Adlin [3].

Matthews et al. [11] focused mainly on designers and user experience professionals who had some training in Personas creation and had done extensive work with Personas using them as described by others [2][3]. These designers had a very positive attitude towards the technique. Those who had done minor use of Personas had a moderate or neutral opinion regarding Personas, and those who had not worked with Personas at all had a negative or indifferent opinion regarding the technique.

The use of the Personas technique in software development processes, e.g., by combining Personas and agile development like XP, has also been explored. In this case, the customer preferred a Persona without a picture, merely describing a job title and maybe a name, but Powell et al. do not support this as it will take away the developers' empathy for the users. Moreover, by using Personas integrated in XP, the developers felt confident to make decisions without involving the onsite customer every time [29].

4) Personas in Practice. An inquiry of design teams in 13 Danish multi national companies report that Personas help keep the focus on user needs instead of what the developers and designers like, and help in gaining an understanding of how the product can create value for end-users [26]. A different study describe how designers are using Personas contrary to the original intended usage; instead of creating Personas on research results, designers tend to base the Personas on their own experiences and thoughts [27]. This will make it even harder to ensure that the right Personas are created to represent the relevant user groups [8]. Problems in application of the Personas technique caused by the mindset of the developers have also been reported [3] [12]. It has been suggested to overcome this by regularly sending information about the Personas to the development team [19][25]. It seems difficult in practice to avoid making stereotypes when creating Personas, and using Personas does not seem to solve the problem that Cooper originally intended to solve [28].

III. Study A: Method

In order to inquire into the usage of Personas we conducted a questionnaire study in 60 software development companies. We chose to focus on a well-defined geographical area in order to allow us to do as complete a survey with as many companies as possible, and thereby achieve a more complete coverage of software companies in that area. The focus on one defined region is that it allows us to establish contact with all companies located in the region. This provides a more complete picture than randomly picking out

companies located in several regions or even countries. We made considerable efforts to identify and contact all companies in the area. The selection of companies would be more random if we had chosen a larger geographical area.

A. Participants

We focused on companies that were developing software, either for internal or external use. We ended up with software companies with the following characteristics:

The company;

- develops software with a graphical user interface (e.g., mobile phones, games, web applications, PC or PDA software).
- develops software for customers or for internal use and is geographically located within the defined geographical area.
- employs more than a single person and it is not a hobby company.

TABLE I. THE NUMBER OF RELEVANT COMPANIES.

	Companies		
Lists used to find companies	***Total number of companies on list***	***Out of scope or gone out of business***	***Relevant companies***
List 1	77	-35	42
List 2	139	-63	76
Linked In	16	0	16
Total			**134**

To obtain a list with as many software development companies as possible we acquired two lists containing software companies located in the chosen region. These lists were from a previous study of companies (List 1) and an industry network (List 2). This was followed by a search on LinkedIn to include companies that only had a smaller development department in the region and had their headquarters located either in another region or in another country. Table I shows the total number of software companies in the region, which were within the scope of this study.

B. Data Collection

We created an online questionnaire using the tool SurveyXact [30]. The first part of the questionnaire was made to gain information about the respondent and his or her place of employment (e.g., job function, business, number of employees in the company and line of business, within software development). The second part was designed to uncover if the respondents knew what a Persona was and what it was used for. The third part was about the use of Personas in the companies. This part was only filled out by the people who answered that they knew of, and worked with, Personas. The questionnaire consisted of 35 questions, but only respondents who knew of and was working with Personas in their current employment got to answer all 35 questions. The questionnaire consisted of both open and closed questions.

The distribution of the questionnaire was done in two ways. First, 43 companies in which we had a known contact person was contacted by phone. Then the remaining 91 of the 134 companies were contacted to acquire a contact person. Eight of these declined to participate and 14 we could not locate a viable phone number or email address. This resulted in 112 emails being sent out with a link to the questionnaire. The recipients were given three weeks to fill out the questionnaire survey. The data collection process resulted in 69 responses in total. Of the 69 respondents nine did not finish the questionnaire, leaving us with 60 complete responses. The nine who did not complete the questionnaire were mainly CEO's in small companies. These respondents mainly stopped filling out the quiestionnaire after entering their personal details.

The responding companies were asked to characterize their main line of business. The distribution is shown in Table II.

TABLE II. THE DISTRIBUTION OF THE COMPANIES AFTER LINE OF BUSINESS.

Characterization of Companies	**Number of Answers**
Software development	44
Design and development	4
Financial services	2
Marketing and advertisement	2
Game development and entertainment	1
Telecom	2
Web development	4
Other line of business	1
Total	**60**

Table II shows that the respondents prevailingly characterize their main line of business as software development.

C. Data Analysis

Data analysis was conducted continuously while the questionnaire was still open for submissions, as suggested by Urquhart [31]. When the questionnaire was closed, the data was updated with the results from the latest incoming questionnaires.

In the questionnaire, we used both open and closed questions. All responses to closed questions were analysed quantitatively. For the open questions, the grounded theory approach, as described by Corbin and Strauss [32], Urquhart [31] and Urquhart et al. [33], was used as analysis method. The aim of grounded theory is described as "building theory, not testing theory" [34]. This means that theory should emerge while the analysis takes place and should not be used to prove an already existing theory.

1) Open Questions: Coding was used to analyse the open questions. One question was: "How would you explain what a Persona is and how it is used?". For this question the fol-

lowing coding categories were assigned: technique (for creating Personas), finding target user group, when in the process the Personas are used and how they are used. Grounded theory coding was not used for other open questions since the respondents mainly answered in very short sentences and they were sent directly to the end of the questionnaire when answering "No", e.g., "Have you ever heard about Personas?" or "Have you ever worked with Personas?" meaning that the number of respondents dropped for every question. As it makes no sense to ask a respondent about their knowledge about the use of Personas if they have already indicated they have never heard about Personas.

2) *Closed Questions:* Statistics was produced directly from the closed questions.

IV. STUDY A: RESULTS

This section presents the results of the questionnaire survey. It is divided into two sub-sections. 'Knowledge about the Personas technique' is referring to the first part of the questionnaire. This subsection reports if the Personas technique has been adopted by the software developing companies in the defined region. The second subsection "The understanding of Personas and their use" is dividing the obstacles towards Personas usage into four main areas.

A. Knowledge about the Personas technique

The results of the questionnaire indicate that 27 out of 60 respondents, or 45%, have heard about Personas. Fourteen respondents out of 60 have worked with Personas. Seven respondents out of 60 are using Personas as a development tool in their current job. This can be seen in Table III.

TABLE III. DISTRIBUTION OF RESPONDENTS AND KNOWLEDGE ABOUT PERSONAS

Knowledge about Personas	**Number of respondents**
Heard about Personas	27 out of 60
Have Worked with Personas	15 out of the 27
Are using Personas in current job	7 out of the 15

Meaning that 11.5% of the responding companies are currently using Personas as a development tool and 55% of the respondents have never heard about the technique.

TABLE IV. DISTRIBUTION OF RESPONDENTS ON COMPANY SIZE.

	Number of Employees				
Number of companies	***1-10***	***11-50***	***51-200***	***200<***	***Total***
Using Personas	1	3	1	2	**7**
Not using Personas	23	16	8	6	**53**
Total	**24**	**19**	**9**	**8**	**60**

The distribution across different sizes of companies is shown in Table IV, showing the number of respondents familiar with Personas.

TABLE V. RESPONDENTS' KNOWLEDGE ABOUT PERSONAS IN COMPANIES THAT DO NOT USE THEM.

	Number of Employees				
Knowledge about Personas	***1-10***	***11-50***	***51-200***	***200<***	***Total***
Never heard about Personas	18	7	6	2	**33**
Heard about Personas, but never used them	4	5	2	2	**13**
Worked with Personas in other employment or while studying	1	2	0	1	**4**
Have used Personas, but stopped	2	1	0	0	**3**
Total	**25**	**15**	**8**	**5**	**53**

In Table V, the 53 responding companies that do not use Personas have been grouped. It shows that 33 respondents have never heard about Personas. Three of the organisations did use Personas at some point but stopped. One respondent stated his organisation used Personas in a project where they collaborated with a group of university students, but did not find the Personas technique useful for other projects. The other two respondents stated that their respective companies stopped using Personas, because they did not find the developed Personas applicable in their line of development. 13 respondents stated they had heard about the Personas technique but had never worked with creating Personas themselves and four respondents had worked with creating Personas in an earlier employment or while studying.

B. Understanding of Personas and their use

An open question in the questionnaire was analysed with coding to reveal all the participating companies' understanding of the term "Persona". "Personas being an imaginary user", were expressed by 22 respondents, e.g., "a fictitious user of the system you are developing". "Personas are used as a validation of the design", were expressed by 17 respondents, e.g., "making sure user needs are met by a given design".

A Persona "being a representation of a larger user segment" was expressed by 13 respondents, e.g., "description of a set of characteristics characterizing a certain group of users' behavioural patterns". Personas "being a tool for making sure to keep the users and their needs in mind all the way through the development process" were recognised by four respondents, e.g., "...the Personas are used as focus points for planning the entire product life cycle". This means that Personas by far are recognised as fictionalised users used as a tool for designing features requested by users and user segments. On the other hand, no more than four respondents expressed that Personas should be used through the entire development cycle. This means that the common idea seems to be that Personas are mainly a tool for identifying some as-

pects of the user group and not so much a tool to be used during the entire development process.

TABLE VI. DISTRIBUTION OF JOB TITLES OF RESPONDENTS.

Job Title of respondents	***Not working with Personas in current employment***	***Currently working with Personas***
CEO, CTO, Owner	12	4
System developer or consultant	11	1
Project, Product or Sales manager	16	0
Business architect, Communication and PR	8	0
UX or Web Designer or Manager	6	2
Total	**53**	**7**

1) Lack of Knowledge (of the technique): Lack of knowledge about the Personas technique seems to be a major obstacle regarding usage of Personas as shown in Table III. The analysis showed that 55% of the respondents had never heard about the concept or technique. Of the respondents who had never heard about Personas, 10 people were CEOs, owners or partners (primarily in micro- or small sized companies), five were managers in IT and three worked as sales managers (all three in medium sized companies). In Table VI, the respondents' job titles have been divided into groups based on whether the company is currently working with Personas, or not. This indicates that the chance of allocating resources to Personas development might be slim. One respondent indicated that the company did not recognise the importance for any communicative tools. "The company has downsized and has eliminated the communications position since it is primarily a production company and they do not really understand the importance of, e.g., Personas, ambassadors, first movers, e.g., or communication in general for that matter". This means that in these companies the knowledge about the Personas technique will not come from management, and even if employees bring the knowledge about Personas into the companies, funding will probably not be allocated. On the other hand, as seen in Table IV, in the seven companies currently working with Personas four respondents was CEO, CTO or owner.

2) Lack of Resources (time and funding): The analysis found that Personas are mainly created if a need has been identified for a specific project and "cutting a corner" when using Personas seems to be the general idea. Some only use Personas to the point that they think it creates value for the customer and thereby, profit for the company. Also, when asked in the survey how much resources were allocated to develop Personas, the general answer was zero.

3) Sparse descriptions: When a Persona is created too superficially the Persona will lack the depth that would normally be the strength of the technique, making the Personas untrustworthy and unusable. This contradicts with what helps making Personas useful tools that lead to better design decisions [2][3][15][16][17]. When a Persona is created with much detail and described as a whole character, and not a stereotype, it will support the design and innovation process. One respondent indicated difficulty in finding a suitable template for the descriptions and that they wanted to create short descriptions instead of detailed character descriptions. "It is hard to find good templates for constructing Personas. We ended up with a few lines in bullets describing each Persona, which could be used as a fast reference. Instead of a large scheme describing lots of details nobody wanted to read anyway". This corresponds with the descriptions of Personas by some respondents answering the questionnaire. These descriptions were quite superficial and did not describe individual Personas but mainly a job role and a use situation.

4) Not integrated in the development: This ties-in with the finding of lacking resources. The superficial Personas are created to be used in the design process. The descriptions are not meant to be used in any other stages of the design process. Furthermore, they are not used to keep reminding neither developers nor designers about the end-user's and their needs. This means that the potential of the Personas technique is not explored.

C. Advantages of using Personas

The respondents currently using Personas described why their companies are using Personas as follows: "*to support the development of a system that is easy to use for all types of users...It is very important for us that the system will be very easy to use, which is why a mapping of the various user groups is important*".

Another respondent stated: "*Internally in the company, Personas are used to communicate characteristics of the customer segments that we want to focus on especially"*. Yet another respondent stated that "*Personas are primarily used for optimizing the product*". These advantages correspond with the advantages identified in the related work section.

V. STUDY B: METHOD

We have conducted a case study about the use of Personas as a development technique in four software development organizations, including if, and how practitioners perveive Personas and how they actually use this technique in practice.

A. Respondents

From Study A software developers were identified, who had different types of experience using Personas as part of the software development process. Four kinds of software developers were identified, whom had different experiences and perceptions in regards to using Personas. One software developer from each category was identified and asked to participate in this study. The four different types are described as follows;

- Wants to start using Personas as a development technique. (R1)
- Has formerly used Personas as a development technique. (R2)

- Is currently using Personas as a development technique. (R3)
- Has knowledge about it but never used it as a development technique. (R4)

R1 – R4 shows which respondent falls under what category.

The respondents were working as software developers or project managers. None of them had any education in user experience. All respondents had worked in the industry for at least ten years and been in their current organization for at least two years. All four interviewees use an agile software development method in their current organisations. All are using SCRUM or an adjusted version of SCRUM.

B. Data Collection

The four interviews were conducted as semi-structured qualitative interviews [35]. The interviews were recorded and later transcribed. Each interview lasted between 22 and 55 minutes. All interviewees were asked about their educational background and their current and previous job functions. Through the interviews the interviewees' knowledge about and previous experiences with the Personas technique was explored.

C. Data Analysis.

All interviews were analysed using grounded theory [32] [33] and open coding with the Dedoose tool (http://www.dedoose.com/). This resulted in the following seven categories;

- Learning to Create Personas
- The Basis for Creating Personas
- Usefulness of Personas
- Strengths of Personas
- Redundancy of Personas
- Weaknesses and Limitations of Personas
- Personas and other techniques

These seven categories were used to categorise the findings.

VI. Study B: Findings

This section presents the findings based on the analysis of the interviews. The findings are divided into seven subsections in accordance with the coding categories.

A. Learning to Create Personas

The respondents learned about the Personas technique in different ways. Their first meeting with Personas seems to mainly have happened by chance. Two respondents describe it this way:

R2: The first time I heard about Personas was at a session at the universitys' humanities department four or five years ago. … Microsoft has created a number of Personas describing the users some years ago. They encourage us, as Microsoft consultants, to use these in our development process.

R1: I have a background as a software developer but in my former employment I worked very closely with user experience designers.

One respondent described coming from a smaller company where he learned about several usability techniques and why it is important to understand and represent the users' in the development process.

None of the respondents learned about Personas and other User-Centered Design or Usability techniques through education.

B. The Basis for Creating Personas

The respondents use different ways of collecting data for the creation of Personas. Yet all of them depend either on information they already have or information their customers have.

R1: If we do not have enough information ourselves to create the Personas we will ask our customers about their usage of the existing systems.

None of the respondents get money or time allocated specifically to gather information about the target user group, which is why they have to make use of the information they already have themselves or they can get from their customers.

Another respondent explained that due to not having a budget for data collecting, he was creating Personas a bit differently than suggested by the literature. He primarily thought about the existing users and the archetypes that were standing out.

R3: We know our users quite well. Our Personas are based on real users, like "can this user understand this?" We use them like Personas archetypes and we do not use Personas formalized. - Unformalized we use Personas quite a lot. Personas are based on the users who are critical towards our system; the people that make noise if they have a problem.

Another respondent described making Personas that were short and without much detail.

R2: To me a Persona does not have to be too detailed in the description of the person.

None of the respondents remembered reading specific literature about Personas. They had mainly learned the do's and don'ts about Personas from others, or from their own experiences.

C. Usefulness of Personas

Personas are considered particularly useful when the developers are missing information about the users and their needs. As all four respondents are employed in companies that use an agile development method, they usually work with an onsite costumer. Personas was found particularly useful if they did not have an onsite customer on a project. The greater the distance between the users, and the designers and developers the more useful Personas are considered to be. One respondent explained that he found Personas very useful as a substitute for onsite customers:

R1: If there is no onsite customer or employee that knows the field we are developing for very well, Personas seems to be very usable. The further the designers and de-

velopers are from the users, the more value Personas can bring to the development process.

Another respondent described Personas as a useful tool if there was a geographical distance between designers and developers. This was meant as Personas could help the developers remember the end-users during the development process. So instead of the design team present to make sure the developers focused on the end-users, Personas could do the same thing, if the Personas was made visible for the developers.

R3: I find Personas useful if the distance between designers and developers is substantial and they are not working side by side all day.

One respondents described that his company does considerable work for the health sector, and they used to have a former nurse employed to help them understand that domain. However, this was no longer an option, so they needed to find new techniques to bring an understanding of the user groups into the development process. He thought Personas could be useful for exactly that.

Another respondent described Personas as useful when developing software solutions for very specific user segments.

R2: We are creating ERP solutions. I feel that Personas are a relevant tool for us. Because we are developing very specific software solutions for our customers.

This respondent also outlined different opinions about the usefulness of Personas and other techniques in regards to User-Centered Design;

R2: One of my colleagues approached me one day and said the following "we live by creating solutions, not drawings." I understand his position but personally I feel that drawing up the organization first can help me understand their needs.

Other respondents described similar experiences of colleagues having different oppinions in regards to using User-Centered Design techniques or Usability theory in regards to software development.

D. Strengths of Personas

The respondents expressed different expectations about the benefits of using Personas in the development process. The respondents were asked to describe situations in which the Personas technique would have been beneficial.

R4: I believe using Personas would have helped us develop a more user-friendly system.

Personas are also perceived as a strong tool for ensuring the software developers keep the end-users in mind during the development process.

R1: Personas can help keeping the developer's focus on the users' needs. Personas will provide the software developer with the ability to understand the users' perspective.

R2: I think that Personas can provide the security for us not developing the wrong system for our user group.

One respondent added that he found Personas especially useful if using a development method like the waterfall method. His argument was that when using the waterfall method the developers have only one possibility to get everything right.

R3: If using the waterfall development method you have to get everything right the first time. When developing agile it is not as critical if we make a mistake, we can change that in the next iteration as a new iteration starts every two weeks.

The respondents find that a strenght of the Personas technique is that it can support the developers in developing software that live up to the users' requirements, and that Personas is especially useful in situations where it is eminent getting it right the first time.

E. Redundancy of Personas

Two respondents stated that Personas are unnecessary if user experience designers or expert users are part of the project team, meaning that the design decisions are not only left to the developers.

R4: Personas are unnecessary when design is not left to the developer but is in place long before the developers begin to create the software.

R3: If you have an employee who is an expert user and knows what the user group need, Personas are unnecessary.

The Personas technique is considered redundant if User Experience Designers or similar are involved in the development process.

F. Weaknesses and Limitations of Personas

The respondents agreed that using Personas incorrectly can have substantial negative impact on software or product development. They also agreed that Personas should not be used if there is insufficient data or if the creators are unfamiliar with Personas.

R2: If the choice you make when creating the Personas is wrong they will work against the design.

Another respondent raised the concern that he felt constrained by some formalized Personas. Every time he was in doubt he went to look at the Persona, but this meant that he felt boxed in, and it stopped him from looking outside of the box.

R3: When using Personas formalized you might be a bit constrained, always going to look at the posters with the Personas [...] To me it works better if I just keep them in my head. Of course our company is not that large anyway so I can just go talk to the developers if I need to change something.

Another respondent had drawn a similar conclusion:

R1: What tends to go wrong in software development is that developers tend to lock on some user requirements pretty early in the process, without documentation, and then describe the entire solution. If the user requirements or the solution change at some point, the developers tend to forget the user and their needs somewhere in the process.

The respondents described using a technique like Personas could be a limitation in regards to the software developers, as the respondents could have a problem changing focus if the requirements changed at some point.

Using Personas requires a certain level of maturity. Another respondent's current organization was not using Personas:

R1:"We are not using the Personas technique at the moment. I have worked with Personas in my last employment and found them very useful. I would like to introduce Personas in my current employment but the company needs to be

at a higher level of maturity before it would make sense. We simply have larger issues at the moment than this".

Using the Personas technique is described as a strenght, but only if the company has reached a certain level of maturity. Personas are perceived as usable if the organisation is unmature.

G. Personas with Other Techniques

The respondents stated that scenarios are very usable in combination with Personas.

R4: Scenarios are often used in combination with Personas.

Workshops and focus groups were also considered useful in combination with creating Personas.

R3: We have a community around our product and we host meetings with user groups, where we meet three times a year and discuss new releases and improvements.

Three respondents described that they are primarily using user stories to document the users' needs. The user stories are described by two respondents as being used instead of developing a specification of requirements.

R3: We use common sense and we are not afraid of making a mistake because it is okay if we do not get it right the first time.

Even though Personas are considered useful the respondents also discribed working agile meaning that correcting errors was not perceived a big deal.

VII. DISCUSSION

In this section, we discuss our results in relation to experiences about Personas reported in the literature, and we compare the findings across the two studies.

The discussion is structured with the following four issues: 1) software developers lack knowledge and understanding of their users, their work, and goals, 2) the Personas technique has been promoted as a strong tool for providing the software developers with a better understanding of the potential users, 3) the use of Personas has been a success, and 4) the Personas technique is not necessarily an incorporated part of the toolbox in the software development industry and the industry might experience problems using Personas.

A. Lack of knowledge and understanding of the users

Software developers lack knowledge and understanding of their users and their needs [5][6]. In many development situations, users do not know what they want, thus, it is the designer's job to find out. Pruitt and Grudin [19] argue that a good design does not come from users, but from designers. This is because users do not really know what they want until they get it. But for this approach to work, the designers need in-depth knowledge of the users and their needs. The aim of Personas is to provide that knowledge.

Among our findings was poor application of the Personas technique in practice. This relates precisely to the point about developers lacking knowledge and understanding of the users, since the Personas descriptions, if applied, are made sparse and only used in a very narrow time frame of the development process. Another finding was that the development of the Personas lacked resources, since none of our respondents had a budget allocated specifically for the Personas development. This is contrary to the related work emphasizing that Personas can lead to better design decisions [2][3][10][15][16][17].

B. Personas can help developers understand users

The Personas technique has been promoted as a strong tool for providing software developers with a better understanding of the potential users [7]. Thus, Personas is presented as a useful technique to keep the developers focused on the users and their needs and give them empathy towards the Personas and the end-users [7][8].

The results from our questionnaire indicate that the most useful aspect of using the Personas technique was that Personas helped the team share a specific and consistent understanding of several, different user groups; which can lead to another advantage of product optimization.

In our case study, we found that the respondents perceived Personas as a technique that supports designing and engineering interactive systems with a focus on the end-users. Matthews et al. [11] found that mainly developers who have been working with Personas are positive in regards to a technique like Personas. We got the same impression from our respondents. Unfortunately, the Personas technique is still suffering from developers considering it unnecessary; e.g., one respondent explained that his colleague told him that creating background material or drawings was a waste of time.

C. Personas used as a successful tool

Several papers conclude the use of Personas has been a success [9][10]. This corresponds with the experiences of our respondents who are using Personas. The tool is described as useful to help developers understand the users and their needs, especially if the system needs to be usable for several different types of end-users. Some respondents using Personas, identified some challenges for creating Personas, e.g., "it can be hard to find templates for creating Personas." another respondent stated that "it is a challenge to map all user groups without asking all customers". These obstacles have to be resolved before Personas can be applied as a useful tool.

In our case study, we found that the practitioners do not use Personas as suggested in the literature. Instead, data is collected before creating Personas and it is mainly collected within their own or the customers' organization, or Personas are created on the basis of real users.

Baird [36] argued that Personas could be developed in a workshop while discovering requirements. One of our respondents described how they used Personas, and hosted meetings with their user group regularly. These meetings were also used to get to know their users and to help get an understanding of the customers' needs.

Personas are primarily considered useful if designers and developers are not working closely together to ensure that the developers understand the intended users and use, or merely as a representation of a user if there is no onsite customer available.

Using Personas has also been described as being risky. If the Personas created are targeting a wrong user group, the

software solution could end up being developed for the wrong users.

Scenarios and user-stories are considered useful in combination with Personas. In particular, user stories have been used to describe user situations and as a requirements specification.

D. Personas are not incorporated in the industry

The Personas technique is not necessarily an incorporated part of the toolbox in the software development industry, and the industry might have problems using Personas [12]. Since only 44% of our respondents have ever heard about the Personas technique and less than 12% have worked with creating Personas, it is fair to say that Personas are not an integrated tool in the software development industry in this region. Also, we found that only four respondents indicated that Personas should be used through the entire development process, meaning that even if Personas are used, they are not necessarily used to their full potential. In companies using Personas, the technique is used mainly to identify types of users or use cases.

The Personas are kept to a minimum and not focused on describing whole characters. As in the related work, we found developers lacking understanding of how to use Personas to gain most from their usage [7][12][19]. The reasons for that could be a combination of several aspects. We found that resources are not allocated specifically for creating Personas, which corresponds with the area of usability in general [5][19][37].

The full potential of Persona usage does not seem to have caught on in the industry. Matthews, Judge and Whittaker [11] found a connection between, on one hand, the perception of Personas and, on the other hand, to what extent the technique was used and, the amount of training the developers had had using Personas.

VIII. Conclusion and Future Work

This paper has reported from a combined questionnaire survey and case study of experiences with creation and use of Personas in software development practice. There are still only few studies of the actual use of Personas in software development practice [1]. The purpose of these studies were to identify both on the overall level and in detail how practitioners in the industry create and use Personas in their development processes.

In the questionnaire study, we explored to what extent Personas were used by software development companies in a defined geographical area and whether they used Personas as proposed in the literature. To accomplish this, we conducted a questionnaire survey with complete responses from 60 software development companies. The study showed that only 7 out of the 60 software development companies used Personas. The results from the questionnaire also uncovered four issues. Lack of knowledge of the technique as such and lack of resources both related to companies not using the Personas technique. Sparse or badly designed descriptions or not being part of the development process both related to poor application, when using the technique.

Our findings are well linked to other studies described in the related work section. Yet our study contributes with a new angle by focusing on making a complete study within a limited geographical area we now have a pretty good idea about if the Personas technique is an integrated tool in software development in this geographical area. We have not been able to find related work that has done a similar study in another country. This means that this paper is the first paper assessing whether and how Personas are used for developing software in the industry.

The main limitation on our results is that we focussed on a defined geographical area. This was necessary to achieve a high level of coverage of all companies in that area. As future work it would be interesting to learn more about the advantages of using Personas. This area still needs further studies even though some advantages have been identified in this paper, also, it would be interesting to learn if companies that do not use Personas are using another tool instead. The number of respondents for the questionnaire survey can also bee seen as a limitation.

We have presented results that are qualitative and based on four developers who have been interviewed in depth. The number of respondents is obviously a limitation of this study; yet only few software companies are using the Personas technique in their development process, so it is very challenging to find even a few respondents with experiences from using the Personas technique. Conducting a qualitative study means that the perspective of the interviewees are in focus. Conducting a study like this obviously requires that the intereviewees are trustworthy and telling the truth from their perspective.

It would be interesting to conduct a more extensive series of interviews with practitioners about their use of Personas and study how that influence the quality of the systems they develop. Also, if there is a correlation between the type of company that uses Personas and the product being developed, and if the use of Personas differs by type of software development company or product being developed. And if the use of Personas differs by the size of the company.

Acknowledgment

We would like to thank the companies and their employees that participated in our questionnaire survey. We would also like to thank the Danish innovation network in Information Technologies, Infinit for providing partial financial support to the research.

References

[1] J. Billestrup, J. Stage, L. Nielsen, and K. S. Nielsen, "Persona usage in software development; Advantages and Obstacles," Proc. of International Conference on Advances in Computer-Human Interactions (ACHI 2014), IARIA, 2014, pp. 359-364.

[2] A. Cooper, "The inmates are running the asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity," Sams Publishers, 1999.

[3] J. Pruitt and T. Adlin, "The Persona Lifecycle: Keeping People in Mind Throughout Product Design," Morgan Kaufman, 2006.

[4] L. Nielsen, "A model for Personas and scenarios creation," Roskilde, Denmark 27th November, 2003, 71.

[5] J. Bak, K. Nguyen, P. Riisgaard, and J. Stage, "Obstacles to usability evaluation in practice: a survey of software

development organizations," Proc. of Nordic Conference on Human-Computer Interaction: Building Bridges (NordiCHI 2008), ACM Press, 2008 pp. 23-32. doi: 10.1145/1463160.1463164

[6] A. Bruun and J. Stage, "Training software development practitioners in usability testing: an assessment acceptance and prioritization," Proceedings of the 24th Australian Computer-Human Interaction Conference (ozCHI 2012), pp. 52-60. doi: 10.1145/2414536.2414545

[7] A. Cooper and R. Reimann, "About face 2.0: The Essentials of Interaction Design," Wiley Publishing, 2003.

[8] L. Nielsen, "Engaging Personas and Narrative Scenarios," Vol 17, PhD Series. Copenhagen: Samfundslitteratur, 2004.

[9] A. Cooper, R. Reimann, and D. Cronin, "About Face 3.0: The Essentials of Interaction Design," Wiley 2007.

[10] A. Dotan, N. Maiden, V. Lichter, and L. Germanovich, "Designing with Only Four People in Mind? - A Case Study of Using Personas to Redesign a Work-Integrated Learning Support System," Procedings of Human-Computer Interaction – INTERACT (INTERACT 2009) pp. 497-509.

[11] T. Matthews, T. Judge, and S. Whittaker, "How Do Designers and User Experience Professionals Actually Perceive and Use Personas?," Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI 2012), ACM Press, pp. 1219-1228. doi: 10.1145/2207676.2208573

[12] Å. Blomquist and M. Arvola, "Personas in action: Ethnography in an Interaction Design Team," Proceedings of Nordic Conference Human-Computer Interaction (NordiCHI 2002), pp. 197-200. doi: 10.1145/572020.572044

[13] L. Nielsen, "Personas – User Focused Design," Human-Computer Interaction. Springer, 2012.

[14] L. Nielsen, "Personas. In The Encyclopedia of Human-Computer Interaction," 2nd Ed., Aarhus, Denmark: The Interaction Design Foundation, http://www.interactiondesign.org/encyclopedia/Personas.html, 2013. 2015.05.30

[15] F. Long, "Real or Imaginary - the Effect of Using Personas in Product Design," IES Conference, Dublin: Irish Ergonomics Review, 2009, pp. 1-10.

[16] J. Ma and C. LeRouge, "Introducing User Profiles and Personas into Information Systems Development," AMCIS. Paper 237, 2007. http://aisel.aisnet.org/amcis2007/237 2015.05.30

[17] T. Miaskiewicza and K. A. Kozarb, "Personas and User-Centered Design: How Can Personas Benefit Product Design Processes?," Design Studies 32, 5. 2011, pp. 417–430. doi:10.1016/j.destud.2011.03.003

[18] A. Cooper, R. Reimann, and D. Cronin, "About Face 3.0: The Essentials of Interaction Design," Wiley 2007.

[19] J. Pruitt and J. Grudin, "Personas: Practice and Theory," Proceedings of the 2003 conference on Designing for user experiences (DUX 2003), pp. 1-15. doi: 10.1145/997078.997089

[20] D. Levin, (2004). "Which Personas are you targeting?," 5 Minute Whitepaper.

[21] W. Quesenbery, "Using Personas: Bringing Users Alive," STC Usability SIG Newsletter-Usability Interface, 2004.

[22] C. N. Chapman and R. Milham, "The Personas' new Clothes: Methodological and Practical Arguments Against a Popular Method," Proceedings of the Human Factors and Ergonomics Society Annual Meeting, October 2006 vol. 50 no. 5 634-636 (HFES 2006), pp. 634-636,. doi: 10.1177/154193120605000503

[23] T. Adlin and J. Pruitt, "The essential Persona lifecycle: Your guide to building and using Personas," Morgan Kaufmann, Burlington, MA, 2010.

[24] E. Friess, "Personas and decision making in the design process: an ethnographic case study," Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI 2012), ACM Press pp. 1209-1218, 2012. doi: 10.1145/2207676.2208572

[25] S. Faily and I. Flechais, "Persona Cases: A Technique for Grounding Personas," Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI 2011), ACM Press, pp. 2267-2270, 2011. doi: 10.1145/1978942.1979274

[26] L. Nielsen, K. S. Nielsen, J. Stage, and J. Billestrup, "Going global with Personas," Procedings of Human-Computer Interaction – INTERACT (INTERACT 20013), pp. 350-357. Springer Berlin Heidelberg. 2013. doi: 10.1007/978-3-642-40498-6_27

[27] Y. Chang, Y. Lim, and E. Stolterman, "Personas: From Theory to Practices," Proceeding of Proc. of Nordic Conference on Human-Computer Interaction: Building Bridges (NordiCHI 2008), pp. 439-442. doi 10.1145/1463160.1463214

[28] P. Turner and S. Turner, "Is stereotyping inevitable when designing with Personas?," Design Studies, 32, 1, 30-44, (2011) doi:10.1016/j.destud.2010.06.002

[29] S. Powell, F. Keenan, and K. McDaid, "Enhancing Agile Requirements Elicitation With Personas," IADIS International Journal on Computer Science and Information Systems, 2(1), 82-95, 2007.

[30] www.survey-xact.com 2015.05.30

[31] C. Urquhart, "Grounded Theory for Qualitative Research: A Practical Guide," Thousand Oaks, California: Sage, 2013.

[32] J. Corbin and A. Strauss, "Basics of Qualitative Research, Techniques and Procedures for Developing Grounded Theory," 3rd edition, Sage Publications, 2008.

[33] C. Urquhart, H. Lehmann, and M. D. Myers, "Putting the Theory Back Into Grounded Theory," Guidelines for grounded theory studies in Information Systems". Info Systems J 20, 2010, pp. 357-381. doi: 10.1111/j.1365-2575.2009.00328.x

[34] S. Pace, "A Grounded Theory of the Flow Experiences of Web Users," Proceedings of International Journal of Human-Computer Studies (IJHES, 2003), pp. 327-363. 2003. doi: doi:10.1016/j.ijhcs.2003.08.005

[35] Kvale, S.: "Interview," København: Hans Reitzel (1997) NordiCHI, 2008, pp. 353-362.

[36] S. Baird, "Using Personas To Discover Requirements," http://philarnold.co.uk/wp-content/uploads/2009/10/User-Personas.pdf (2002) 2015.05.30

[37] D. Svanæs and J. Gulliksen, "Understanding the Context of Design – Towards Tactical User Centered Design," Proceedings of Nordic Conference oon Human-Computer Interaction: Building Bridges (NordiCHI, 2008), pp. 353-362. doi: 10.1145/1463160.1463199

A Feedback-Controlled Adaptive Middleware for Near-Time Bulk Data Processing

Martin Swientek
Paul Dowland

School of Computing and Mathematics
Plymouth University
Plymouth, UK
e-mail: {martin.swientek, p.dowland}@plymouth.ac.uk

Bernhard Humm
Udo Bleimann

Department of Computer Science
University of Applied Sciences Darmstadt
Darmstadt, Germany
e-mail: {bernhard.humm, udo.bleimann}@h-da.de

***Abstract*—The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change over time. In this article, the concept of a middleware is introduced that adapts its processing type fluently between batch processing and single-event processing using a feedback-control loop. By adjusting the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios. The proposed middleware concept has been implemented with a research prototype and has been evaluated. The results of the evaluation show that the concept is viable and is able to optimize the end-to-end latency of a system for bulk data processing.**

Keywords–adaptive middleware; message aggregation; latency; throughput.

I. INTRODUCTION

This article extends previous work in [1]. Enterprise Systems like customer-billing systems or financial transaction systems are required to process large volumes of data in a fixed period of time. For example, a billing system for a large telecommunication provider has to process more than 1 million bills per day. Those systems are increasingly required to also provide near-time processing of data to support new service offerings.

Traditionally, enterprise systems for bulk data processing are implemented as batch processing systems [2]. Batch processing delivers high throughput but cannot provide near-time processing of data, that is, the end-to-end latency of such a system is high. End-to-end latency refers to the period of time that it takes for a business process, implemented by multiple subsystems, to process a single business event. For example, consider the following billing system of a telecommunications provider:

- Customers are billed once per month
- Customers are partitioned in 30 billing groups
- The billing system processes 1 billing group per day, running 24h under full load.

In this case, the mean time for a call event to be billed by the billing system is 1/2 month. That is, the mean end-to-end latency of this system is 1/2 month.

A. An Example: Billing Systems for Telecommunications Carriers

An example of a system for bulk data processing is a billing system of a telecommunications carrier. A billing system is a distributed system consisting of several sub components that process the different billing sub processes like mediation, rating, billing and presentment (see Figure 1).

The performance requirements of such a billing system are high. It has to process more than 1 million records per hour and the whole batch run needs to be finished in a limited timeframe to comply with service level agreements with the print service provider. Since delayed invoicing causes direct loss of cash, it has to be ensured that the bill arrives at the customer on time.

Figure 1. Billing process

B. Near-Time Processing of Bulk Data

A new requirement for systems for bulk data processing is near-time processing. Near-time processing aims to reduce the end-to-end latency of a business process, that is, the time that is spent between the occurrence of an event and the end of its processing. In case of a billing system, it is the time between the user making a call and the complete processing of this call including mediation, rating, billing and presentment.

The need for near-time charging and billing for telecommunications carriers is induced by market forces, such as the increased advent of mobile data usage and real-time data services [3]. Carriers want to offer new products and services that require real-time or near-time charging and billing. Customers want more transparency, for example, to set their own limits and alerts for their data usage, which is currently only possible for pre-paid accounts. Currently, a common approach for carriers is to operate different platforms for real-time billing of pre-paid accounts and traditional batch-oriented billing for post-paid accounts. To reduce costs, carriers aim to converge these different platforms.

A lower end-to-end latency can be achieved by using single-event processing, for example, by utilizing a message-oriented middleware for the integration of the services that

form the enterprise system. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. Therefore, message-based processing is usually not considered for building a system for bulk data processing requiring high throughput.

The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. A system is therefore either optimized for low latency or high maximum throughput. These requirements are not fixed and can change during the lifespan of a system, either anticipated or not anticipated.

Additionally, enterprise systems often need to handle load peaks that occur infrequently. For example, think of a billing system with moderate load over most of the time, but there are certain events with very high load such as New Year's Eve. Most of the time, a low end-to-end latency of the system is preferable when the system faces moderate load. During the peak load, it is more important that the system can handle the load at all. A low end-to-end latency is not as important as an optimized maximum throughput in this situation.

In this article, a solution to this problem is proposed:

- The concept of a middleware is presented that is able to adapt its processing type fluently between batch processing and single-event processing. By adjusting the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios.
- A prototype has been built to evaluate the concepts of the adaptive middleware.
- A performance evaluation has been conducted using this prototype to evaluate the proposed concept of the adaptive middleware.

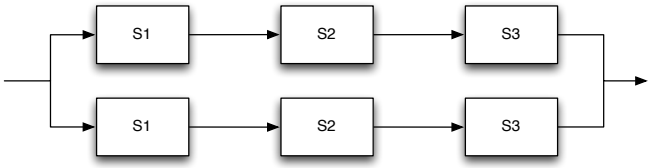
This article extends the adaptive middleware concept, which has been presented in [1]. It adds a discussion of its underlying concepts and design aspects, that should be considered when implementing such an adaptive middleware for near-time processing of bulk data. In addition, it describes the prototype implementation of the middleware concept and presents the results of the evalution of the propposed approach, as well as its limitations.

The remainder of this article is organized as follows. Section II defines the considered type of system and the terms throughput and latency. Section III gives an overview of other work related to this research. The concept, components and design aspects of the adaptive middleware are presented in Section IV through VI. Section VII describes the prototype system that has been build to evaluate the proposed concepts. The evaluation of the prototype system is presented in Section VIII. Section IX describes the limitations of this research. Finally, Section X concludes the paper and gives and outlook to further research.

II. Background

We consider a distributed system for bulk data processing consisting of several subsystems running on different nodes that together form a processing chain, that is, the output of subsystem S1 is the input of the next subsystem S2 and so on (see Figure 2a).

(a) Single processing line



(b) Parallel processing lines

Figure 2. A system consisting of several subsystems forming a processing chain

To facilitate parallel processing, the system can consist of several lines of subsystems with data being distributed among each line. For simplification, a system with a single processing line is considered in the remainder of this article.

We discuss two processing types for this kind of system, batch processing and message-based processing.

A. Batch processing

The traditional operation paradigm of a system for bulk data processing is batch processing (see Figure 3). A batch processing system is an application that processes bulk data without user interaction. Input and output data is usually organized in records using a file- or database-based interface. In the case of a file-based interface, the application reads a record from the input file, processes it and writes the record to the output file.

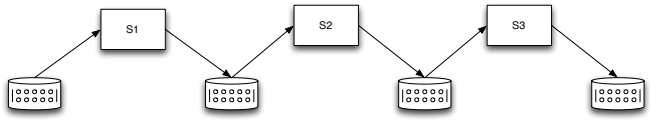


Figure 3. Batch processing

B. Message-base processing

Messaging facilitates the integration of heterogeneous applications using asynchronous communication. Applications are communicating with each other by sending messages (see Figure 4). A messaging server or message-oriented middleware handles the asynchronous exchange of messages including an appropriate transaction control [4].

Figure 4. Message-based processing

Message-based systems are able to provide near-time processing of data due to their lower latency compared with batch processing systems. The advantage of a lower latency comes with a performance cost in regard to a lower maximum throughput because of the additional overhead for each processed message. Every message needs, amongst others, to be serialized and deserialized, mapped between different protocols and routed to the appropriate receiving system.

C. End-to-end Latency vs. Maximum Throughput

Throughput and latency are performance metrics of a system. We are using the following definitions of maximum throughput and latency in this article:

- **Maximum Throughput**
  The number of events the system is able to process in a fixed timeframe.
- **End-To-End Latency**
  The period of time between the occurrence of an event and its processing. End-to-end latency refers to the total latency of a complete business process implemented by multiple subsystems. The remainder of this article focusses on end-to-end latency using the general term latency as an abbreviation.

Latency and maximum throughput are opposed to each other given a fixed amount of processing resources. High maximum throughput, as provided by batch processing, leads to high latency, which impedes near-time processing. On the other hand, low latency, as provided by a message-based system, cannot provide the maximum throughput needed for bulk data processing because of the additional overhead for each processed event.

III. Related Work

This section gives an overview of work related to the research presented in this article. It discusses performance optimizations in the context of transport optimization, middleware optimizations and message batching.

The proposed middleware for high-performance near-time processing of bulk data adjusts the data granularity itself at runtime. Work on middleware discusses different approaches for self-adjustment and self-awareness of middleware, which can be classified as adaptive or reflective middleware.

Automatic scaling of server instances is another approach to handle infrequent load spikes. Additionally, the section gives a brief overview of feedback-control of computing systems.

Research on messaging middleware currently focusses on Enterprise Service Bus (ESB) infrastructure. An ESB is an integration platform that combines messaging, web services, data transformation and intelligent routing to connect multiple heterogeneous services [5]. It is a common middleware to implement the integration layer of an Service Oriented Architecture (SOA) and is available in numerous commercial and open-source packages.

A. Transport Optimization

Most of the work that aims to optimize the performance of service-oriented systems is done in the area of Web Services since it is a common technology to implement a SOA.

In particular, various approaches have been proposed to optimize the performance of SOAP, the standard protocol for Web Service communication. This includes approaches for optimizing the processing of SOAP messages (cf. [6] [7] [8]), compression of SOAP messages (cf. [9] [10]) and caching (cf. [11] [12]). A survey of the current approaches to improve the performance of SOAP can be found in [13].

[14] proposes an approach to transfer bulk data between web services per File Transfer Protocol (FTP). The SOAP messages transferred between the web services would only contain the necessary details how to download the corresponding data from an FTP server since this protocol is optimized for transferring huge files. This approach solves the technical aspect of efficiently transferring the input and output data but does not pose any solutions how to implement loose coupling and how to integrate heterogeneous technologies, the fundamental means of an SOA to improve the flexibility of an application landscape.

Data-Grey-Box Web Services are an approach to transfer bulk data between Web Services [15]. Instead of transferring the data wrapped in SOAP messages, it is transferred using an external data layer. For example, when using database systems as a data layer, this facilitates the use of special data transfer methods such ETL (Extract, Transform, Load) to transport the data between the database of the service requestor and the database of the Web service. The data transfer is transparent for both service participants in this case. The approach includes an extension of the Web service interface with properties describing the data aspects. Compared to the SOAP approach, the authors measured a speedup of up to 16 using their proposed approach. To allow the composition and execution of Data-Grey-Box Web services, [16] developed BPEL data transitions to explicitly specify data flows in BPEL processes.

[17] proposes three tuning strategies to improve the performance of Java Messaging Service (JMS) for cloud-based applications.

1) When using persistent mode for reliable messaging the storage block size should be matched with the message size to maximize message throughput.
2) Applying distributed persistent stores by configuring multiple JMS destinations to achieve parallel processing
3) Choosing appropriate storage profiles such as RAID-1

In contrast, the optimization approach presented in this thesis is aimed at the integration layer of messaging system, which allows further optimizations, such as dynamic message batching and message routing.

B. Middleware Optimizations

Some research has been done to add real-time capabilities to ESB or messaging middleware. [18] proposes an architecture for a real-time messaging middleware based on an ESB. It consists of an event scheduler, a JMS-like API and a communication subsystem. While fulfilling real-time

requirements, the middleware also supports already deployed infrastructure.

In their survey [19], the authors describe a real-time ESB model by extending the Java Business Integration (JBI) specification with semantics for priority and time restrictions and modules for flow control and bandwidth allocation. The proposed system is able to dynamically allocate bandwidth according to business requirements.

MPAB (Massively Parallel Application Bus) is an ESB-oriented messaging bus used for the integration of business applications [20]. The main principle of MPAB is to fragment an application into parallel software processing units, called SPU. Every SPU is connected to an Application Bus Multiplexor (ABM) through an interface called Application Bus Terminal (ABT). The Application Bus Multiplexor manages the resources shared across the host system and communicates with other ABM using TCP/IP. The Application Bus Terminal contains all the resources needed by SPU to communicate with its ABM. A performance evaluation of MPAB shows that it achieves a lower response time compared to the open source ESBs Fuse, Mule and Petals.

Tempo is a real-time messaging system written in Java that can be used on either a real-time or non-real-time architecture [21]. The authors, Bauer et al., state that existing messaging systems are designed for transactional processing and therefore not appropriate for applications with with stringent requirements of low latency with high throughput. The main principle of Tempo is to use an independent queuing system for each topic. Resources are partitioned between these queueing systems by a messaging scheduler using a time-base credit scheduling mechanism. In a test environment, Tempo is able to process more than 100,000 messages per second with a maximum latency of less than 120 milliseconds.

In contrast to these approaches, the approach presented in this thesis is based on a standard middleware and can be used with several integration technologies, such as JMS or SOAP.

C. Message Batching

Aggregating or batching of messages is a common approach for optimizing performance and has been applied to several domains. TCP Nagle's algorithm is a well-known example of this approach [22].

Message batching for optimizing the throughput of Total Ordering Protocols (TOP) have first been investigated by [23]. In their work, the authors have compared the throughput and latency of four different Total Ordering Protocols. They conclude that "batching messages is the most important optimization a protocol can offer".

[24] extends the work of [23] with a policy for varying the batch level automatically, based on dynamic estimates of the optimal batch level.

[25] presents a mechanism for self-tuning the batching level of Sequencer-based Total Order Broadcast Protocols (STOB), that combines analytical modeling an Reinforcement Learning (RL) techniques.

[26] proposes a self-tuning algorithm based on extremum seeking optimization principles for controlling the batching level of a Total Order Broadcast algorithm. It uses multiple instances of extremum seeking optimizers, each instance is associated with a distinct value of batching b and learns the optimal waiting time for a batch of size b.

[27] describes two generic adaptive batching schemes for replicated servers, which adapt their batching level automatically and immediately according to the current communication load, without any explicit monitoring of the system.

The approach presented in this research applies the concept of dynamic message batching to minimize the end-to-end latency of a message-based system for bulk data processing.

D. Self-Adaptive Middleware

[28] argues that "the most adequate level and natural locus for applying adaption is at the middleware level". Adaption at the operating system level is platform-dependent and changes at this level affect every application running on the same node. On the other hand, adaption at application level assigns the responsibility to the developer and is also not reusable.

[29] proposes an adaptive, general-purpose runtime infrastructure for effective resource management of the infrastructure. Their approach is comprised of three components:

1) dynamic performance prediction
2) adaptive intra-site performance management
3) adaptive inter-site resource management

The runtime infrastructure is able to choose from a set of performance predictions for a given service and to dynamically choose the most appropriate prediction over time by using the prediction history of the service.

AutoGlobe [30] provides a platform for adaptive resource management comprised of

1) Static resource management
2) Dynamic resource management
3) Adaptive control of Service Level Agreements (SLA)

Static resource management optimizes the allocation of services to computing resources and is based on on automatically detected service utilisation patterns. Dynamic resource management uses a fuzzy controller to handle exceptional situations at runtime. The Adaptive control of Service Level Agreements (SLAs) schedules service requests depending on their SLA agreement.

The coBRA framework proposed by [31] is an approach to replace service implementations at runtime as a foundation for self-adaptive applications. The framework facilitates the replacement of software components to switch the implementation of a service with the interface of the service staying the same.

DREAM (Dynamic Reflective Asynchronous Middleware) [32] is a component-based framework for the construction of reflective Message-Oriented Middleware. Reflective middleware "refers to the use of a causally connected self-presentation to support the inspection and adaption of the middleware system" [33]. DREAM is based on FRACTAL, a generic component framework and supports various asynchronous communication paradigms such as message passing,

event-reaction and publish/subscribe. It facilitates the construction and configuration of Message-Oriented Middleware from a library of components such as message queues, filters, routers and aggregators, which can be assembled either at deploy-time or runtime.

E. Adaption in Service-Oriented Architectures

Several adaption methods have been proposed in the context of service-based applications. In their survey [34], the authors describe the following adaption methods:

- **Adaption by Dynamic Service Binding**
  This adaption method relies on the ability to select and dynamically substitute services at run-time or at deployment-time. Services are selected in such a way that the adaption requirements are satisfied in the best possible way.
- **Quality of Service (QoS)-Driven Adaption of Service Compositions**
  The goal of this adaption approach is to select the best set of services available at run-time, under consideration of process constraints, end-user preferences and the execution context.
- **Adaption of Service Interfaces and Protocols**
  The goal of this adaption approach is to mediate between two services with different signatures, interfaces and protocols. This includes signature-based adaption, ontology-based adaption or behavior-based adaption.

F. Adaptive ESB

Research on messaging middleware currently focusses on ESB infrastructure. An ESB is an integration platform that combines messaging, web services, data transformation and intelligent routing to connect multiple heterogeneous services [5]. It is a common middleware to implement the integration layer of an Service Oriented Architecture (SOA) and is available in numerous commercial and open-source packages.

Several work has been done to extend the static service composition and routing features of standard ESB implementations with dynamic capabilities decided at run-time, such as dynamic service composition [35], routing [36] [37] [38] and load balancing [39].

The DRESR (Dynamic Reconfigurable ESB Service Routing), proposed by [36], allows the routing table to be changed dynamically at run-time based on service selection preferences, such as response time. It defines mechanisms to test and evaluate the availability and performance of a service and to select services based on its testing results and historical data.

[38] proposes a framework for content-based intelligent routing. It evaluates the service availability and selects services based on its content and properties.

[39] proposes a load balancing mechanism that distributes requests to services of the same *service type*, having the same function and signature, and enables the dynamic selection of the target service.

Work to manage and improve the QoS of ESB and service-based systems in general is mainly focussed on dynamic service composition and service selection based on monitored QoS metrics such as throughput, availability and response time [40].

[41] proposes an adaptive ESB infrastructure to address QoS issues in service-based systems, which provides adaption strategies for response time degradation and service saturation, such as invoking an equivalent service, using previously stored information, distributing requests to equivalent services, load balancing and deferring service requests.

In contrast to these solutions, the approach presented in this article uses dynamic message aggregation and message routing as adaption mechanism to optimize the end-to-end latency of messaging system for different load scenarios.

G. Automatic Scaling

A different solution to handle infrequent load spikes is to automatically instantiate additional server instances, as provided by current Platform as a Service (PaaS) offerings such as Amazon EC2 [42] or Google App Engine [43]. While scaling is a common approach to improve the performance of a system, it also leads to additional operational and possible license costs. Additionally, it is often difficult to scale certain components or external dependencies of the system, such as databases or external services. Of course, the approach presented in this article can be combined with these auto-scaling approaches.

H. Feedback-control of Computing Systems

Feedback-control has been applied to several different domains of computing systems since the early 1990s, including data networks, operating systems, middleware, multimedia and power management (cf. [44]). Feedback-control of middleware systems include application servers, such as the Apache http-Server, database management systems, such as IBM Universal Database Server, and e-mail servers, such as the IBM Lotus Domino Server. [44] describes 3 basic control problems in this context:

- Enforcing service level agreements
- Regulate resource utilization
- Optimize the system configuration

Additionally, feedback-control has been applied recently to web environments, such as web servers and web services, application servers, including data flow control in J2EE servers, Repair Management in J2EE servers and improving the performance of J2EE servers and cloud environments (cf. [45]).

The *Adaptive Middleware* presented in this article utilizes a closed-feedback loop to control the aggregation size of the processed messages, depending on the current load of the system to minimize the end-to-end latency of the system. This is a novel approach that has not previously been investigated.

IV. Middleware Concepts

The adaptive middleware is based on the following core concepts: (1) message aggregation, (2) message routing, and (3) monitoring and control.

A. Message Aggregation

Message aggregation or batching of messages is the main feature of the adaptive middleware to provide a high maximum throughput. The aggregation of messages has the following goals:

- To decrease the overhead for each processed message
- To facilitate optimized processing

There are different options to aggregate messages, which can be implemented by the Aggregator:

- **No correlation**: Messages are aggregated in the order in which they are read from the input message queue. In this case, an optimized processing is not simply possible.
- **Technical correlation:** Messages are aggregated by their technical properties, for example, by message size or message format.
- **Business correlation**: Messages are aggregated by business rules, for example, by customer segments or product segments.

In [1], a static aggregation size has been used to optimize the latency and the throughput of a system. This is not feasible for real systems, since the the latency and throughput also depends on the load of the system. Therefore, a dynamic aggregation size depending on the current load of the system is needed.

B. Message Routing

The goal of the message routing is to route the message aggregate to the appropriate service, which is either optimized for batch or single event processing, to allow for an optimized processing. Message routing depends on how messages are aggregated. Table I shows the different strategies of message routing.

TABLE I
STRATEGIES FOR MESSAGE ROUTING

Routing Strategy	Examples	Description
Technical routing	Aggregation size	Routing is based on the technical properties of a message aggregate.
Content-based routing	Customer segments (e.g. business customers or private customers)	Routing is based on the content of the message aggregate, that is, what type of messages are aggregated.

With high levels of message aggregation, it is not preferred to send the aggregated message payload itself over the message bus using Java Messaging Service (JMS) or SOAP. Instead, the message only contains a pointer to the data payload, which is transferred using File Transfer Protocol (FTP) or a shared database.

Message routing can be static or dynamic:

- **Static routing:**
  Static routing uses static routing rules, that are not changed automatically.
- **Dynamic routing:**
  Dynamic routing adjusts the routing rules automatically at run-time, for example, depending on QoS properties of services. For example, see [36], [37] or [38].

C. Monitoring and Control

In order to optimize the end-to-end latency of the system, the middleware needs to constantly monitor the load of the system and control the aggregation size accordingly (see Figure 5).

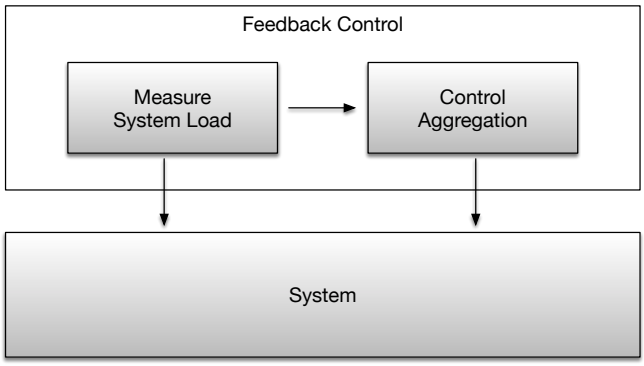


Figure 5. Monitoring and Control

If the current load of the system is low, the aggregation size should be small to provide a low end-to-end latency of the system. If the current load of the system is high, the aggregation size should be high to provide a high maximum throughput of the system.

To control the level of message aggregation at runtime, the adaptive middleware uses a closed feedback loop as shown in Figure 6, with the following properties:

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.

Ultimately, we want to control the average end-to-end latency depending on the current load of the system. The change of queue size seems to be an appropriate quantity because it can be directly measured without a lag at each sampling interval, unlike for example, the average end-to-end latency.

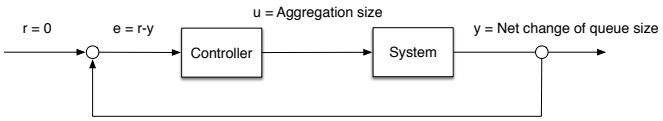


Figure 6. Feedback loop to control the aggregation size

V. MIDDLEWARE COMPONENTS

Figure 7 shows the components of the middleware, that are based on the Enterprise Integration Patterns described by [46]. A description of these components can be found in Table II.

VI. DESIGN ASPECTS

This section describes aspects that should be taken into account when designing an adaptive system for bulk data processing.

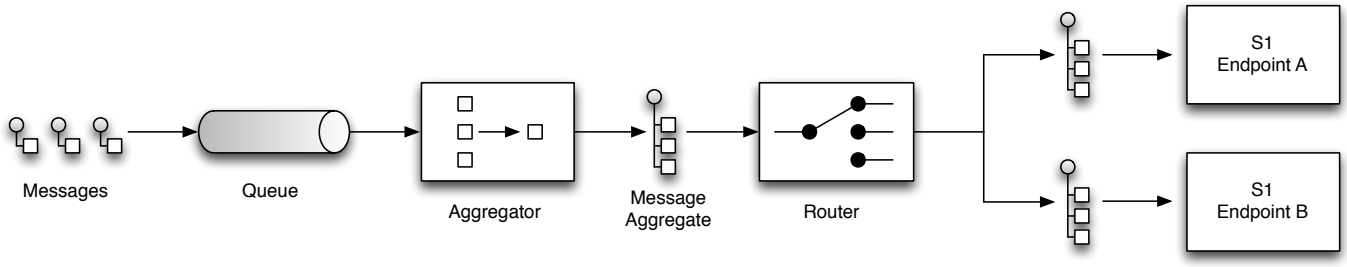


Figure 7. Middleware components

TABLE II
COMPONENTS OF THE ADAPTIVE MIDDLEWARE. WE ARE USING THE NOTATION DEFINED BY [46]

Symbol	Component	Description
	Message	A single message representing a business event.
	Message Aggregate	A set of messages aggregated by the Aggregator component.
	Queue	Storage component which stores messages using the FIFO principle.
	Aggregator	Stateful filter which stores correlated messages until a set of messages is complete and sends this set to the next processing stage in the messaging route.
	Router	Routes messages to the appropriate service endpoint.
Service Endpoint	Service Endpoint	Represents a business service.

A. Service Design

The services that implement the business functionality of the system need to be explicitly designed to support the runtime adaption between single-event and batch processing.

There are different options for the design of these services:

- Single service interface with distinct operations for single and batch processing
  - The service provides different distinct operations for high and low aggregation sizes with optimized implementations for batch and single-event processing. The decision which operation should be called is done by the message router. It is generally not possible to use different transports for different aggregation sizes.
- Single service interface with a single operation for both single and batch processing
  - The service provides a single operation that is called for all aggregation sizes. The decision which optimization should be used is done by the service implementation. It is not possible to use different transports for different aggregation sizes.
- Multiple service interfaces for single and batch processing (or different aggregation sizes)
  - The logical business service is described by distinct service interfaces which contain operations for either batch processing or single-event processing. The decision which operation should be called is done by the message router. It is possible to use different transports for different aggregation sizes.

The choice of service design relates to where you want to have the logic for the message routing for optimized processing. With a single service offering distinct operations for single-event and batch processing, as well as with distinct service for each processing style, the message router decides which service endpoint should be called. In contrast, using a single service with a single operation for both processing styles, the service itself is responsible for choosing the appropriate processing strategy. Using a different integration type for each processing style is not possible in this case.

Listing 1 shows the interface of a service offering different operations for batch processing (line 6) and single-event processing (line 10).

B. Integration and Transports

The integration architecture defines the technologies that are used to integrate the business services. In general, different integration styles with different transports are used for batch processing and single-event processing, which needs to be taken into account when designing an adaptive system for bulk data processing.

When using high aggegration sizes, it is not feasible to use the same transports as with low aggregation sizes. Large messages should not be transferred over the messaging system. Instead, a file based transport using FTP or database-based integration should be used. When using a messaging system, the payload of large messages should not be transported over the messaging system. For example, by implementing the *Claim Check* Enterprise Integration Pattern (EIP) (cf. [46]).

Additionally, the technical data format should be considered.

The concrete threshold between low and high aggregation sizes depends on the integration architecture and implementation of the system, such as the integration architecture and the deployed messaging system.

Listing 1. Java interface of a web service offering different operations for single and batch processing.

```
1  @WebService
2  @SOAPBinding(style=Style.DOCUMENT, use=Use.LITERAL, parameterStyle=ParameterStyle.WRAPPED)
3  public interface RatingPortType {
4    @WebMethod(operationName="processCallDetails")
5    @WebResult(name="costedEvents")
6    public Costedevents processCallDetails(@WebParam(name="callDetailRecords") SimpleCDRs
         callDetailRecords) throws ProcessingException, Exception;
7
8    @WebMethod(operationName="processCallDetail")
9    @WebResult(name="costedEvent")
10   public Costedevent processCallDetail(@WebParam(name="simpleCDR") SimpleCDR callDetailRecord)
         throws ProcessingException, Exception;
11 }
```

TABLE III
TRANSPORT OPTIONS FOR HIGH AND LOW AGGREGATION SIZES

Aggregation Size	Transport Options
High	• Database • File-based (e.g. FTP) • Claim Check EIP
Low	• JMS • SOAP

The choice of the appropriate integration transport for a service is implicitly implemented by the message router (see Section IV-B).

C. Error Handling

Message aggregation has also an impact on the handling of errors that occur during the processing. Depending on the cause of the error, there are two common types of errors:

- **Technical errors**
  Technical errors are errors caused by technical reasons, for example, an external system is not available or does not respond within a certain timeout or the processed message has an invalid format.
- **Business errors**
  Business errors are caused by violation of business rules, for example, a call detail record contains a tariff that is no longer valid.

The following points should be taken into account, when designing the error handling for an adaptive system for bulk data processing:

- Write erroneous messages to an error queue for later processing.
- Use multiple queues for different types of errors, for example, distinct queues for technical and business errors to allow different strategies for handling them. Some type of errors can be fixed automatically, for example, an error that is caused by an outage of an external system, while other errors need to be fixed manually.
- If the erroneous messages is part of an aggregated message, it should be extracted from the aggregate to prevent the whole aggregate from beeing written to the error queue, especially when using high aggregation sizes.

D. Controller Design

There are several approaches for the implementation of feedback-control systems. [44] describes two major steps:

1) modeling the dynamics of the system
2) developing a control system

There are different approaches that are used in practice to model the dynamics of a system [47]:

- Empirical approach using curve fitting to create a model of the system
- Black-box modeling
- Modeling using stochastic approaches, especially queuing theory
- Modeling using special purpose representations, for example, the first principles analysis

For practical reasons, the following approach has been taken in this research:

1) Define the control problem
2) Define the input and output variables of the system
3) Measure the dynamics of the system
4) Develop the control system

1) Control Problem: The control problem is defined as follows:

- Minimize the end-to-end latency of the system by controlling the message aggregation size.
- The aggregation size used by the messaging system should depend on the current load of the system.
- When the system faces high load, the aggregation size should be increased to maximize the maximum throughput of the system.
- When the system faces low load, the aggregation size should be decreased to minimize the end-to-end latency of the system.

2) Input/Output Signals: [48] describes the following criteria for selecting input control signals:

- **Availability**
  It should be possible to influence the control input directly and immediately.

- **Responsiveness**
  The system should respond quickly to a change of the input signal. Inputs whose effect is subject to latency or delays should be avoided when possible.
- **Granularity**
  It should be possible to adjust the control input in small increments. If the control input can only be adjusted in fixed increments, then it could be necessary to consider this in the controller or actuator implementation.
- **Directionality**
  How does the control input impact the control output? Does an increased control input result in increased or decreased output?

Additionally, the following criteria should be considered for selecting output control signals:

- **Availability**
  The quantity must be observable without gaps and delays.
- **Relevance**
  The output signal should be relevant for the behavior of the system that should be controlled.
- **Responsiveness**
  The output signal should reflect changes of the state of the system quickly without lags and delays.
- **Smoothness**
  The output signal should be smooth and does not need to be filtered.

With regard to these criteria, the following input and output control signals have been chosen

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.

3) Control Strategy: We use a simple non-linear control strategy that could be implemented as follows (cf. [48]):

- When the tracking error is positive, increase the aggregation size by 1
- Do nothing when the tracking error is zero.
- Periodically decrease the aggregation size to test if a smaller queue size is able to handle the load.

VII. Prototype Implementation

To evaluate the proposed concepts of the adaptive middleware, a prototype of a billing system has been implemented using Apache Camel [49] as the messaging middleware.

Figure 8 shows the architecture of the prototype. It consists of three nodes, the billing route, mediation service and rating service. The billing route implements the main flow of the application. It is responsible for reading messages from the billing queue, extracting the payload, calling the mediation and rating service and writing the processed messages to the database. The mediation service is a webservice representing the mediation component. It is a SOAP service implemented using Apache CXF and runs inside an Apache Tomcat container. The same applies to the rating service, representing the rating component.

TABLE IV
Technologies and frameworks used for the implementation of the prototypes

Language	Java 1.6
Dependency Injection	Spring 3.0.7
Persistence API	OpenJPA (JPA 2.0) 2.1.1
Database	MySQL 5.5.24
Logging	Logback 1.0.1
Test	JUnit 4.7
Batch Framework	Spring Batch 2.1.8
Messaging Middleware	Apache Camel 2.10.0
Other Frameworks	Joda-Time, Apache Commons

The prototypes are implemented with Java 1.6 using Java Persistence API (JPA) for the data-access layer and a MySQL database. See Table IV for complete list of technologies and frameworks used for the implementation of the prototypes.

The prototype performs the following steps:

1) The message is read from the billing queue using JMS. The queue is hosted by an Apache ActiveMQ instance.
2) The message is unmarshalled using JAXB.
3) The *Mediation service* is called by the CXF endpoint of the billing route.
4) The response of the *Mediation webservice*, the normalized call detail record, is unmarshalled.
5) The *Rating service* is called by the CXF endpoint of the billing route.
6) The response of the *Rating webservice*, that is the costed event, is unmarshalled.
7) The costed event is written to the *Costed Events Database*.

The feedback-control loop of the prototype is implemented by the following components:

- **Performance Monitor**
  The *Performance Monitor* manages the feedback-control loop by periodically calling the *Sensor* and updating the *Controller*. Additionally, it calculates the current throughput and end-to-end latency of the system.
- **Sensor**
  The *Sensor* is responsible for getting the current size of the message queue using Java Management Extensions (JMX).
- **Controller**
  The *Controller* calculates the new value for the aggregation size based on the setpoint and the current error.
- **Actuator**
  The *Actuator* is responsible for setting the new aggregation size of the *Aggregator* calculated by the *Controller*.

A. Aggregator

The *Aggregator* is configured to dynamically use the aggregation size (*completionSize*) set by a message header, as shown in Listing 2 (line 2). This message header is set by the *Actuator* (see Section VII-B3), which is controlled by the *Controller* (see Section VII-B2).

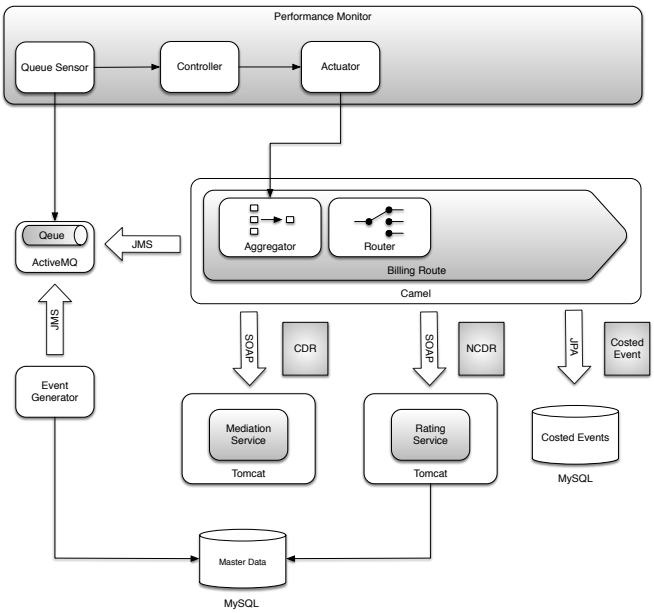


Figure 8. Components of the prototype system

Listing 2. Aggregator configuration in definition of BillingRoute

```
1  .aggregate(constant(true), new
       UsageEventsAggrationStrategy())
2    .completionSize(header(completionSizeHeader)
       )
3    .completionTimeout(completionTimeout)
4    .parallelProcessing()
```

B. Feedback-Control Loop

Figure 9 shows the components of the feedback-control loop.

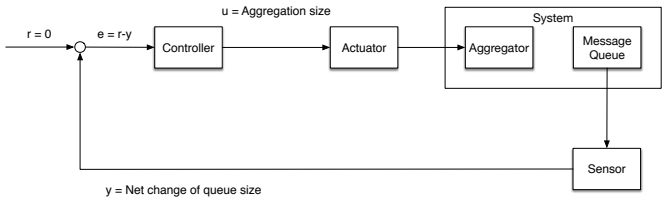


Figure 9. Components of the feedback-control loop

1) Sensor: The *JmxSensor* implements the *Sensor* interface (see Figure 10). It reads the current length of the input queue of the *ActiveMQ* server instance using JMX.

2) Controller: A *Controller* has to implement the *Controller* interface. The following implementations of the *Controller* interface have been implemented (see Figure 11):

- **BasicController**
  Implements a generic controller. The control strategy is provided by an implementation of the *ControllerStrategy*.
- **TestController**
  A controller used for testing the static behavior of the system.

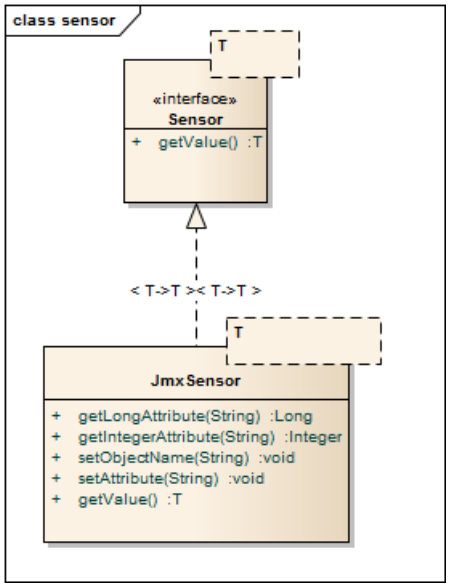


Figure 10. UML classdiagram showing the sensor classes

The strategy of the controller is implemented by a controller strategy which implements the *ControllerStrategy* interface.

Figure 12 shows the available implementations of the *ControllerStrategy*.

Listing 3 shows the implementation of the simple control strategy, as described in Section VI-D3:

- If the queue size increases, increase the aggregation size (line 10-13).
- Otherwise, do not change the aggregation size (line 22).
- Periodically decrease the aggregation size by one (line 17-20).

The controller uses two different timers depending on the previous action.

3) Actuator: The *AggregateSizeActuator* is responsible for setting the aggregation size of the *Aggregator* and is controlled by the *Controller* (see Figure 13).

The *AggregateSizeActuator* implements the *Actuator* interface. It sets the aggregation size (*completionSize*) by setting a specific header in the currently processed message.

4) Performance Monitor: The *Performance Monitor* manages the feedback-control loop by periodically calling the *Sensor* and updating the *Controller*. Additionally, it calculates the current throughput and end-to-end latency of the system using the *StatisticsService* (see Figure 14).

C. Load Generator

The *Load Generator* is used to generate the system load by generating events (Call Detail Records (CDRs)) and writing them to the input message queue of the system. It is implemented as a stand-alone Java program using a command-line interface.

The *DataGenerator* uses a *Poisson Process* to simulate the load of the system, which is commonly used to model

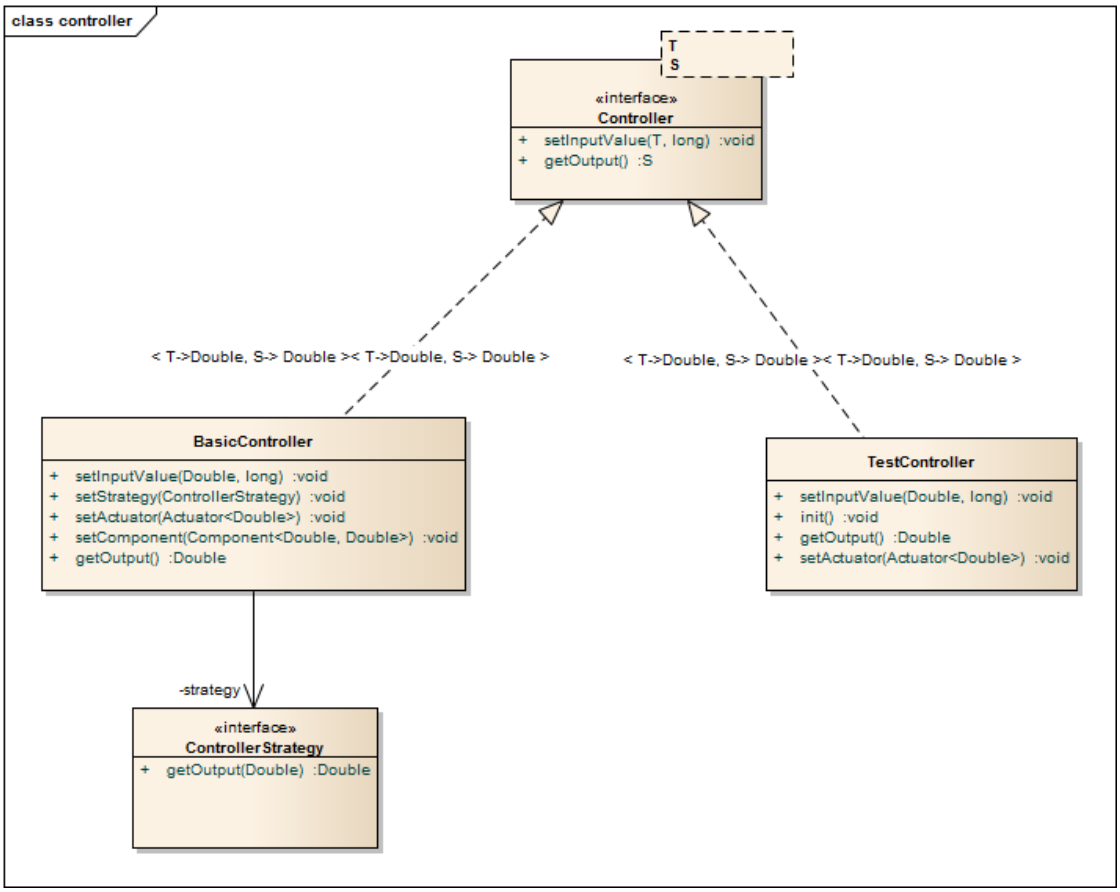


Figure 11. UML classdiagram showing the controller classes

events that occur continuously and independently of each other with exponentially distributed inter-arrival times, e.g. to model requests on a web server [50] or telephone calls [51].

VIII. Evaluation

The prototype described in the previous section has been used to evaluate the general feasibility of the adaptive middleware.

A. Test Environment

The tests have been run on a development machine to decrease the development-build-deploy cycle, as described in Table V.

TABLE V
Test environment

Memory	3 GiB
CPU	Intel Core i5 M520 @ 2,40 GHz
Architecture	32-bit
Disk Drive	150 GB SSD
Operating System	Windows 7
Database	MySQL 5.5.24
Messaging Middleware	Apache ActiveMQ 5.6.0

B. Test Design

[52] defines a set of properties, that should be considered when designing feedback-control systems for computing systems, called the SASO properties (**S**table, **A**ccurate, **S**ettling times, **O**vershoot):

- **Stability**
  The system should provide a bounded output for any bounded input.
- **Accuracy**
  The measured output of the control system should converge to the reference input.
- **Settling time**
  The system should converge quickly to its steady state.
- **Overshoot**
  The system should achieve its objectives in a manner that does not overshoot.

C. Static Tests

To test the relationship between the input and output variables of the control-loop, aggregation size and change of queue size, the following static tests have been performed:

- The *TestController* has been configured to periodically increase the aggregation size after 100 time steps (1 time step equals 1 second).

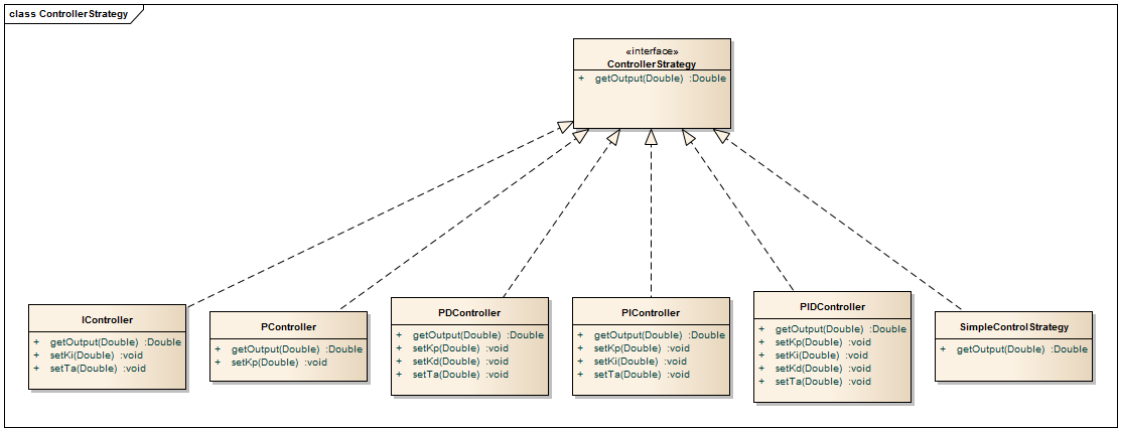


Figure 12. UML classdiagram showing the controller strategy classes

Listing 3. Implementation of the simple control strategy

```
1  public class SimpleControlStrategy
2    implements ControllerStrategy {
3
4    @Value("${simpleController.period1}")
5    private int period1;
6    @Value("${simpleController.period2}")
7    private int period2;
8    private int timer = 0;
9
10   public Double getOutput(Double error) {
11     if (error > 0) {
12       timer = period1;
13       return +1.0;
14     }
15
16     timer--;
17
18     if (timer == 0) {
19       timer = period2;
20       return -1.0;
21     }
22     return 0.0;
23   }
24 }
```

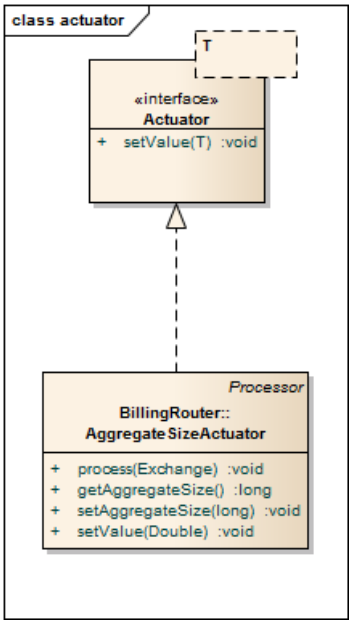


Figure 13. UML classdiagram showing the actuator classes

- The test has been repeated with different load of the system, that is, using different arrival rates for the *DataGenerator*.

Figure 15 shows the queue size of the system in relationship to the aggregation size, for different arrival rates.

- The system is not able to handle the load with an $aggregationsize < 5$ and an $arrivalrate = 50$. With an $aggregationsize \geq 5$, the system is able to process the events faster than they occur.
- With an $arrivalrate = 100$, the system is not able to handle the load with an $aggregationsize < 15$. With an $aggregationsize \geq 15$, the system is able to process the events faster than they occur.
- With an $arrivalrate = 150$, the system is not able to handle the load with an $aggregationsize < 25$. With an $aggregationsize \geq 25$, the system is able process the events faster than they occur.

The change of queue size between each time step is shown in Figure 16.

D. Step Test

To measure the dynamic response of the system, the following step test has been performed:

- The *TestController* has been configured to increase the aggregation size from 1 to 50.
- Messages occur with an arrival rate of 150.

Figure 17 shows the result of the step test:

- With an aggregation size of 1, the system is not able to handle the load. The queue length is constantly increasing.
- When the aggregation size is set to 50 at timestep 100, the queue size is directly decreased, without a noticeable delay.

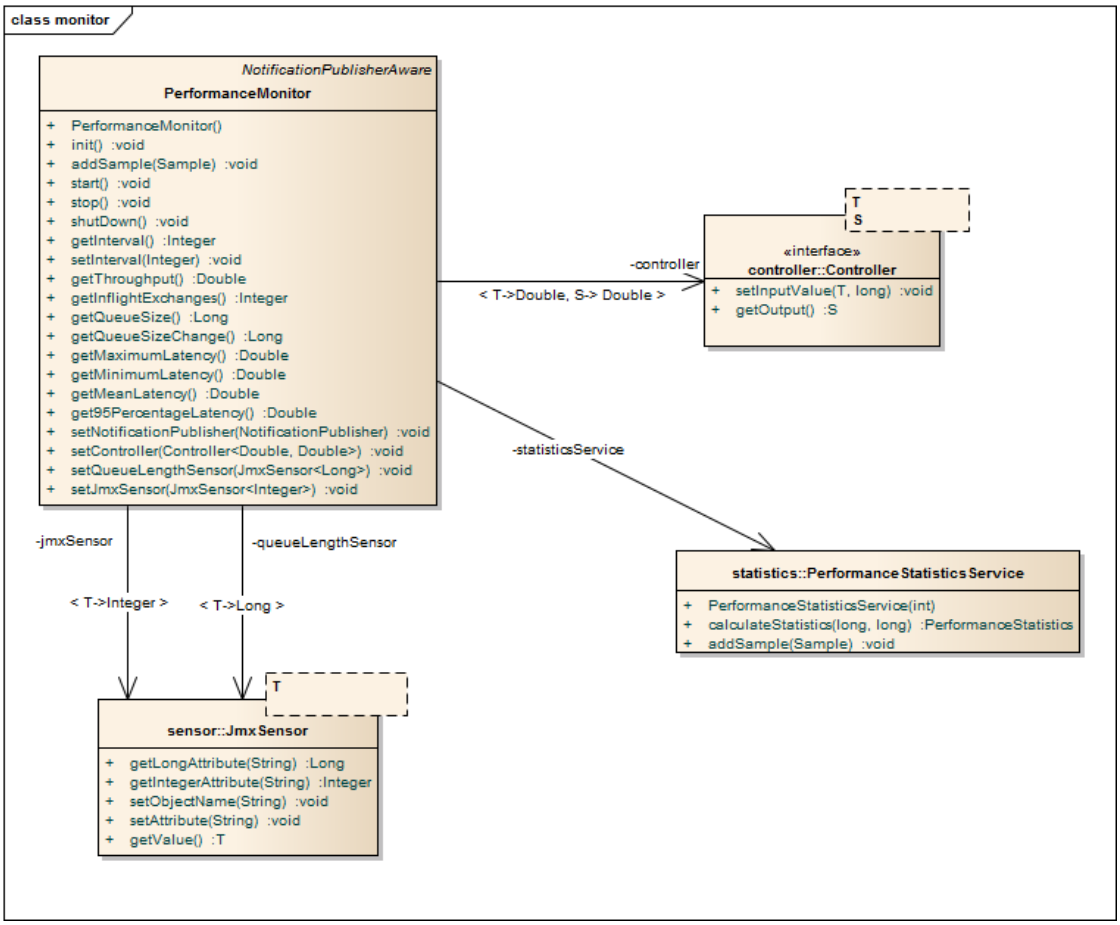


Figure 14. UML classdiagram showing the *PerformanceMonitor*

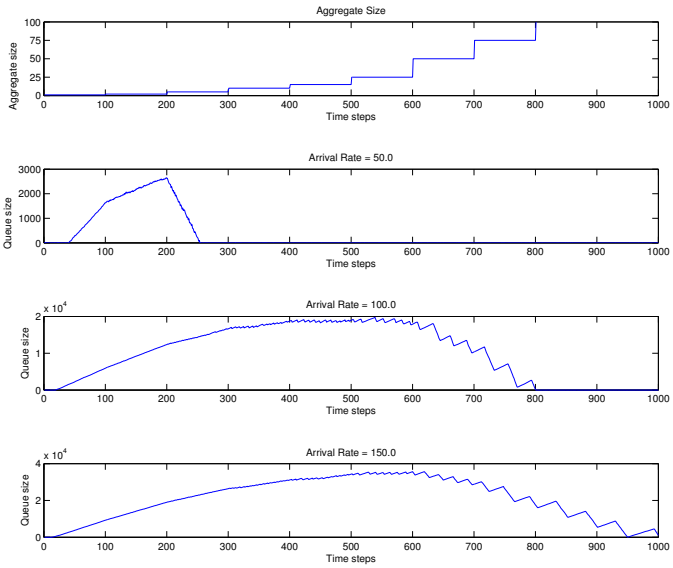


Figure 15. Static test: queue sizes

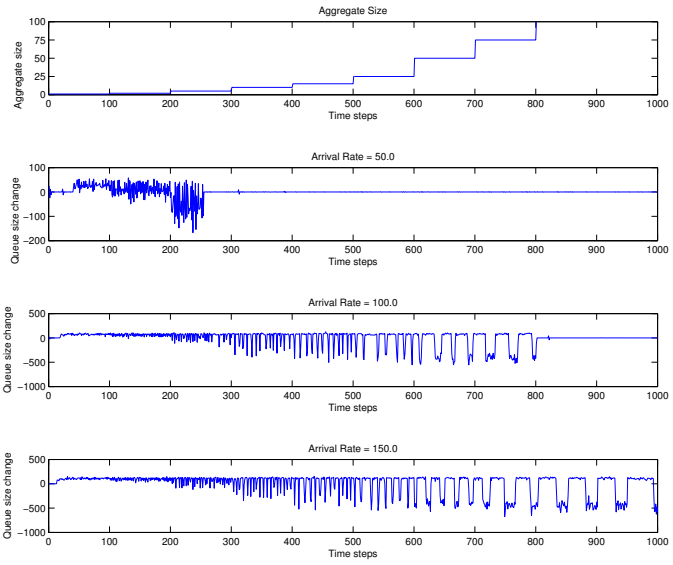


Figure 16. Static test: queue size changes

E. Controller Tests

The following test has been performed to evaluate the performance of the *Simple Controller*:

- Events are generated with an $arrival\ rate = 50.0$ for 100 time steps.
- At $timestep = 100$, the arrival rate is set to 150.0 for another 100 time steps.

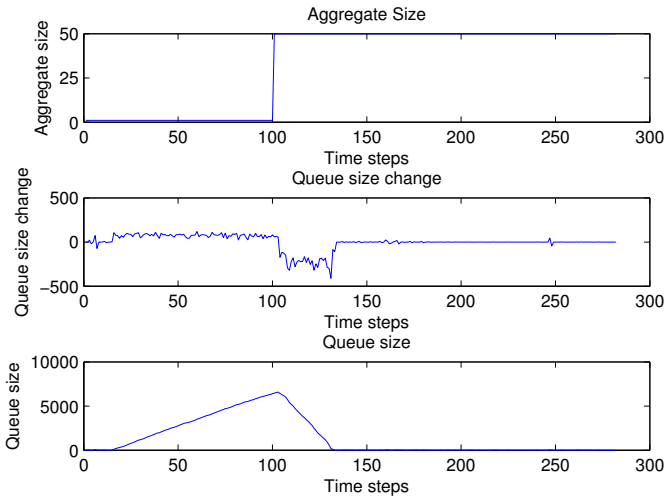


Figure 17. Step test

- At $timestep = 200$, the arrival rate is set back to 50.0.

Figure 18 shows the results of the test using the *Simple Control* strategy:

- The controller is reasonably able to control the size of the queue. At $timestep = 100$, it increases the aggregate size to a maximum value of 36.
- At $timestep = 200$, the controller starts to decrease the aggregation size. At $timestep = 375$, the aggregation size is back at 3.

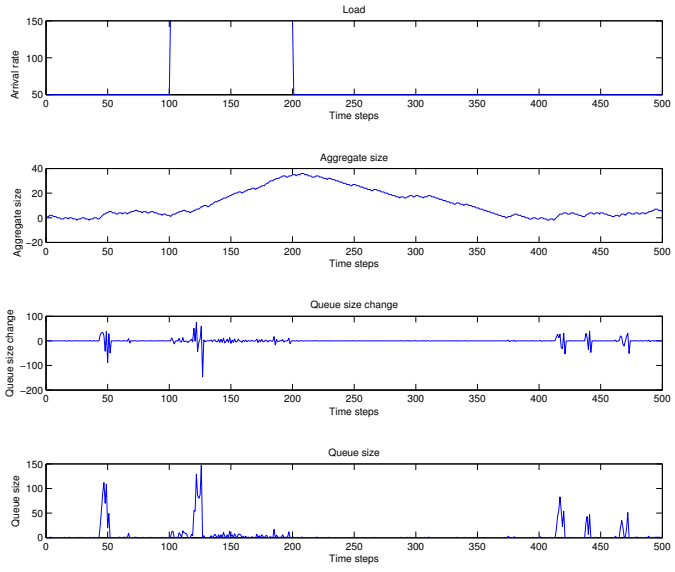


Figure 18. Simple control strategy

F. Results

Summarizing the results of the evaluation, the proposed concept for the adaptive middleware is a viable solution to optimize the end-to-end latency of data processing system. The results show that using a closed-feedback loop is a feasible technique for implementing the dynamic control of the aggregation size. Using the queue size change to measure the system load is also shown to be appropriate.

IX. Limitations

The research presented in this article has some limitations, that are summarized below:

- The services that implement the business functionality of the system need to be explicitly designed to support the run-time adaption between single-event and batch processing, as described in Section VI-A. Therefore, existing services need to be changed in order to be integrated into the system. This can pose a problem when using off-the-shelf services or Software as a Service (SaaS). The integration of such services has not been considered in this research.
- The services integrated by the prototype do not implement any further optimizations for batch processing. They use the same implementation for batch and single-event processing. Thus, the impact of batch optimizations has not been investigated. This was not necessary to show the performance improvements of message aggregation on the maximum throughput of the messaging prototype.
- The adaption mechanisms of the *Adaptive Middleware* only uses message aggregation and message routing, depending on the aggregation size. Other mechanisms such as dynamic service composition and selection and load balancing have not been investigated.
- The prototype of the *Adaptive Middleware* only uses a single message queue, the integrated services are called synchronous, using a request/response pattern. This design was chosen, to simplify the dynamics of the system. Thus, the impact of using multiple message queues has been investigated in the evaluation.
- The impact of different controller architectures has not been exhaustively analyzed and researched. Only two controller architectures have been implemented and evaluated. Other controller designs, such as fuzzy control, have not been investigated. Additionally, a formal analyzation of the feedback-control system has not been conducted, for example, by creating a model of the system. Instead, an empirical approach has been taken to evaluate the viability of the proposed solution.

X. Conclusion and Further Research

In this section, a novel concept of middleware for near-time processing of bulk data has been presented, which is able to adapt itself to changing load scenarios by fluently shifting the processing type between single event and batch processing. The middleware uses a closed feedback loop to control the end-to-end latency of the system by adjusting the level of message aggregation depending on the current load of the system. Determined by the aggregation size of a message, the middleware routes a message to appropriate service endpoints, which are optimized for either single-event or batch processing.

Additionally, several design aspects have been described that should be taken into account when designing and implementing an adaptive system for bulk data processing, such

as how to design the service interfaces, the integration and transport mechanisms, the error-handling and controller design.

The solution is based on standard middleware, messaging technologies and standards and fully preserves the benefits of an SOA and messaging middleware, such as:

- Loose coupling
- Remote communication
- Platform language Integration
- Asynchronous communication
- Reliable Communication

To evaluate the proposed middleware concepts, a prototype system has been developed. The tests show that the proposed middleware solution is viable and is able to optimize the end-to-end latency of a bulk data processing system for different load scenarios.

The next steps of this research are to further analyze the dynamics of the system and to optimize the controller.

During the implementation of the prototype of the adaptive middleware, it became apparent that the design and implementation of such a system differs from common approaches to implement enterprise software systems. Further research addresses a conceptual framework that guides the design, implementation and operation of a system for bulk data processing based on the adaptive middleware.

REFERENCES

[1] M. Swientek, B. Humm, U. Bleimann, and P. Dowland, "An Adaptive Middleware for Near-Time Processing of Bulk Data," in ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications, Venice, Italy, May 2014, pp. 37–41.

[2] J. Fleck, "A distributed near real-time billing environment," in Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99, 1999, pp. 142–148.

[3] J. Cryderman, "Is Real-Time Billing and Charging a Necessity?" Pipeline, vol. 7, no. 11, 2011.

[4] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch, Enterprise Application Integration: Grundlagen, Konzepte, Entwurfsmuster, Praxisbeispiele. Elsevier, Spektrum, Akad. Verl., 2006.

[5] D. Chappell, Enterprise Service Bus. Sebastopol, CA, USA: O'Reilly Media, Inc., 2004.

[6] N. Abu-Ghazaleh and M. J. Lewis, "Differential Deserialization for Optimized SOAP Performance," in SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing. Washington, DC, USA: IEEE Computer Society, 2005, p. 21.

[7] T. Suzumura, T. Takase, and M. Tatsubori, "Optimizing Web Services Performance by Differential Deserialization," in ICWS '05: Proceedings of the IEEE International Conference on Web Services. Washington, DC, USA: IEEE Computer Society, 2005, pp. 185–192.

[8] A. Ng, "Optimising Web Services Performance with Table Driven XML," in ASWEC '06: Proceedings of the Australian Software Engineering Conference. Washington, DC, USA: IEEE Computer Society, 2006, pp. 100–112.

[9] J. C. Estrella, M. J. Santana, R. H. C. Santana, and F. J. Monaco, "Real-Time Compression of SOAP Messages in a SOA Environment," in SIGDOC '08: Proceedings of the 26th annual ACM international conference on Design of communication. New York, NY, USA: ACM, 2008, pp. 163–168.

[10] A. Ng, P. Greenfield, and S. Chen, "A Study of the Impact of Compression and Binary Encoding on SOAP Performance," in Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005), 2005.

[11] D. Andresen, D. Sexton, K. Devaram, and V. Ranganath, "LYE: a high-performance caching SOAP implementation," in Proceedings of the 2004 International Conference on Parallel Processing (ICPP-2004), 2004, pp. 143–150.

[12] K. Devaram and D. Andresen, "SOAP optimization via parameterized client-side caching," in Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003), 2003, pp. 785–790.

[13] J. Tekli, E. Damiani, R. Chbeir, and G. Gianini, "Soap processing performance and enhancement," Services Computing, IEEE Transactions on, vol. 5, no. 3, 2012, pp. 387–403.

[14] T. Wichaiwong and C. Jaruskulchai, "A Simple Approach to Optimize Web Services' Performance," in NWESP '07: Proceedings of the Third International Conference on Next Generation Web Services Practices. Washington, DC, USA: IEEE Computer Society, 2007, pp. 43–48.

[15] D. Habich, S. Richly, and M. Grasselt, "Data-Grey-Box Web Services in Data-Centric Environments," in IEEE International Conference on Web Services, 2007. ICWS 2007, 2007, pp. 976–983.

[16] D. Habich, S. Richly, S. Preissler, M. Grasselt, W. Lehner, and A. Maier, "BPEL-DT – Data-Aware Extension of BPEL to Support Data-Intensive Service Applications," Emerging Web Services Technology, vol. 2, 2007, pp. 111–128.

[17] Z. Zhuang and Y.-M. Chen, "Optimizing jms performance for cloud-based application servers," in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 828–835.

[18] L. Garces-Erice, "Building an enterprise service bus for real-time soa: A messaging middleware stack," in Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International, vol. 2, 2009, pp. 79–84.

[19] C. Xia and S. Song, "Research on real-time esb and its application in regional medical information exchange platform," in Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on, vol. 4, 2011, pp. 1933–1937.

[20] R. Benosman, Y. Albrieux, and K. Barkaoui, "Performance evaluation of a massively parallel esb-oriented architecture," in Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE International Conference on, 2012, pp. 1–4.

[21] D. Bauer, L. Garces-Erice, S. Rooney, and P. Scotton, "Toward scalable real-time messaging," IBM Systems Journal, vol. 47, no. 2, 2008, pp. 237–250.

[22] J. Nagle, "Congestion control in ip/tcp internetworks," SIGCOMM Comput. Commun. Rev., vol. 14, no. 4, Oct. 1984, pp. 11–17. [Online]. Available: http://doi.acm.org/10.1145/1024908.1024910

[23] R. Friedman and R. V. Renesse, "Packing messages as a tool for boosting the performance of total ordering protocls," in Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, ser. HPDC '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 233–.

[24] A. Bartoli, C. Calabrese, M. Prica, E. A. Di Muro, and A. Montresor, "Adaptive Message Packing for Group Communication Systems," 2003, pp. 912–925.

[25] P. Romano and M. Leonetti, "Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning," in Computing, Networking and Communications (ICNC), 2012 International Conference on, Jan 2012, pp. 786–792.

[26] D. Didona, D. Carnevale, S. Galeani, and P. Romano, "An extremum seeking algorithm for message batching in total order protocols," in Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on, Sept 2012, pp. 89–98.

[27] R. Friedman and E. Hadad, "Adaptive batching for replicated servers," in Reliable Distributed Systems, 2006. SRDS '06. 25th IEEE Symposium on, 2006, pp. 311–320.

[28] H. A. Duran-Limon, G. S. Blair, and G. Coulson, "Adaptive Resource Management in Middleware: A Survey," IEEE Distributed Systems Online, vol. 5, no. 7, 2004, p. 1. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1018100&type=external&coll=ACM&dl=GUIDE&CFID=59338606&CFTOKEN=18253396

[29] B.-D. Lee, J. B. Weissman, and Y.-K. Nam, "Adaptive middleware supporting scalable performance for high-end network services," J. Netw. Comput. Appl., vol. 32, no. 3, 2009, pp. 510–524.

[30] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive Quality of Service Management for Enterprise Services," ACM Trans. Web, vol. 2, no. 1, 2008, pp. 1–46.

[31] F. Irmert, T. Fischer, and K. Meyer-Wegener, "Runtime Adaptation in a Service-Oriented Component Model," in SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. New York, NY, USA: ACM, 2008, pp. 97–104.

[32] M. Leclercq, V. Quéma, and J.-B. Stefani, "DREAM: a Component Framework for the Construction of Resource-Aware, Reconfigurable MOMs," in ARM '04: Proceedings of the 3rd workshop on Adaptive and reflective middleware. New York, NY, USA: ACM, 2004, pp. 250–255.

[33] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The Case for Reflective Middleware," Commun. ACM, vol. 45, no. 6, 2002, pp. 33–38.

[34] R. Kazhamiakin, S. Benbernou, L. Baresi, P. Plebani, M. Uhlig, and O. Barais, "Adaptation of service-based systems," in Service Research Challenges and Solutions for the Future Internet, ser. Lecture Notes in Computer Science, M. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, Eds. Springer Berlin Heidelberg, 2010, vol. 6500, pp. 117–156. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-17599-2_5

[35] S.-H. Chang, H. J. La, J. S. Bae, W. Y. Jeon, and S. D. Kim, "Design of a dynamic composition handler for esb-based services," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 287–294.

[36] X. Bai, J. Xie, B. Chen, and S. Xiao, "Dresr: Dynamic routing in enterprise service bus," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 528–531.

[37] B. Wu, S. Liu, and L. Wu, "Dynamic reliable service routing in enterprise service bus," in Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE, Dec 2008, pp. 349–354.

[38] G. Ziyaeva, E. Choi, and D. Min, "Content-based intelligent routing and message processing in enterprise service bus," in Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on, Aug 2008, pp. 245–249.

[39] A. Jongtaveesataporn and S. Takada, "Enhancing enterprise service bus capability for load balancing," W. Trans. on Comp., vol. 9, no. 3, Mar. 2010, pp. 299–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1852392.1852401

[40] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," Software Engineering, IEEE Transactions on, vol. 37, no. 3, May 2011, pp. 387–409.

[41] L. González and R. Ruggia, "Addressing qos issues in service based systems through an adaptive esb infrastructure," in Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, ser. MW4SOC '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/2093185.2093189

[42] "Amazon ec2 auto scaling," http://aws.amazon.com/autoscaling, [retrieved: March 2014].

[43] "Auto scaling on the google cloud platform," https://cloud.google.com/developers/articles/auto-scaling-on-the-google-cloud-platform, [retrieved: March 2014].

[44] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, Feedback Control of Computing Systems. John Wiley & Sons, 2004.

[45] R. K. Gullapalli, C. Muthusamy, and V. Babu, "Control systems application in java based enterprise and cloud environments–a survey," Journal of ACSA, 2011.

[46] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[47] J. L. Hellerstein, "Challenges in control engineering of computing systems," in American Control Conference, 2004. Proceedings of the 2004, 2004, pp. 1970–1979.

[48] P. K. Janert, Feedback Control for Computer Systems. O'Reilly Media, Inc., 2013.

[49] "Apache Camel," http://camel.apache.org, 2014, [retrieved: July 2014].

[50] M. F. Arlitt and C. L. Williamson, "Internet Web servers: workload characterization and performance implications," IEEE/ACM Transactions on Networking (TON), vol. 5, no. 5, Oct. 1997, pp. 631–645.

[51] D. Willkomm, S. Machiraju, J. Bolot, and A. Wolisz, "Primary user behavior in cellular networks and implications for dynamic spectrum access," Communications Magazine, IEEE, vol. 47, no. 3, March 2009, pp. 88–95.

[52] T. Abdelzaher, Y. Diao, J. Hellerstein, C. Lu, and X. Zhu, "Introduction to Control Theory And Its Application to Computing Systems," in Performance Modeling and Engineering, Z. Liu and C. Xia, Eds. Springer US, 2008, pp. 185–215–215.

A Study on Transport and Load in a Grid-based Manufacturing System

Leo van Moergestel, Erik Puik, Daniël Telgen
Department of Computer science
HU Utrecht University of Applied Sciences
Utrecht, the Netherlands
Email: {leo.vanmoergestel, erik.puik, daniel.telgen}@hu.nl

John-Jules Meyer
Intelligent systems group
Utrecht University
Utrecht, the Netherlands
Email: J.J.C.Meyer@uu.nl

***Abstract*—Standard mass-production is a well-known manufacturing concept. To make small quantities or even single items of a product according to user specifications at an affordable price, alternative agile production paradigms should be investigated and developed. The system presented in this article is based on a grid of cheap reconfigurable production units, called equiplets. A grid of these equiplets is capable to produce a variety of different products in parallel at an affordable price. The underlying agent-based software for this system is responsible for the agile manufacturing. An important aspect of this type of manufacturing is the transport of the products along the available equiplets. This transport of the products from equiplet to equiplet is quite different from standard production. Every product can have its own unique path along the equiplets. In this article several topologies are discussed and investigated. Also, the planning and scheduling in relation to the transport constraints is subject of this study. Some possibilities of realization are discussed and simulations are used to generate results with the focus on efficiency and usability for different topologies and layouts of the grid and its internal transport system. Closely related with this problem is the scheduling of the production in the grid. A discussion about the maximum achievable load on the production grid and its relation with the transport system is also included.**

Keywords-Multiagent-based manufacturing; Flexible transport.

I. INTRODUCTION

In standard batch processing the movement of products is mostly based on a pipeline. Though batch processing is a very good solution for high volume production, it is not apt for agile manufacturing when different products at small quantities are to be produced by the production equipment. This article describes an agile and flexible production system, where the production machines are placed in a grid. Products are not following a single path, but different paths can be used in parallel, leading to parallel manufacturing of different products. The grid arrangement of production machines reduces the average path when products move along their own possibly unique paths within the grid during the production. To move the products around during production, the ways the production machines are interconnected should be investigated to find an affordable and good solution. An important aspect will also be the amount of products in the grid during production, because too many products will result in failures in the scheduling of the production. The investigation about transport and the amount of products in the grid, resulting in the total load of the grid, are the motivation and purpose of this article. The goal is to investigate the effect of different interconnection possibilities to the average production path and to see how the grid behaves under load. The work in this article is based on a paper presented at the Intelli 2014 conference [1] and other previous work. The design and implementation of the production platforms and the idea to build a production grid can be found in Puik [2]. In Moergestel [3] the idea of using agent technology as a software infrastructure is presented. Two types of agents play a major role in the production: a product agent, responsible for production of a product and an agent responsible for performing certain production steps on a production machine. Another publication by Moergestel [4] is dedicated to the production scheduling for the grid production system. The rest of this paper is organised as follows: In Section II of this article related work will be discussed. Section III will explain grid manufacturing in more detail, followed by Section IV about transport in the grid. Section V introduces the software tools built. The results are presented and discussed in Section VI. Finally, a conclusion where the results are summarized will end the article.

II. RELATED WORK

Using agent technology in industrial production is not new though still not widely accepted. Important work in this field has already been done. Paolucci and Sacile [5] give an extensive overview of what has been done in this field. Their work focuses on simulation as well as production scheduling and control [6]. The main purpose to use agents in [5] is agile production and making complex production tasks possible by using a multi-agent system. Agents are also introduced to deliver a flexible and scalable alternative for manufacturing execution systems (MES) for small production companies. The roles of the agents in this overview are quite diverse. In simulations agents play the role of active entities in the production. In production scheduling and control agents support or replace human operators. Agent technology is used in parts or subsystems of the manufacturing process. On the contrary, we based the manufacturing process as a whole on agent technology. In our case a co-design of hardware and software was the basis.

Bussmann and Jennings [7][8] used an approach that compares to our approach. The system they describe introduced three types of agents, a workpiece agent, a machine agent and

a switch agent. Some characteristics of their solution are:

- The production system is a production line that is built for a certain product. This design is based on redundant production machinery and focuses on production availability and a minimum of downtime in the production process. Our system is a grid and is capable to produce many different products in parallel;
- The roles of the agents in this approach are different from our approach. The workpiece agent sends an invitation to bid for its current task to all machine agents. The machine agents issue bids to the workpiece agent. The workpiece agent chooses the best bid or tries again. In our system, the negotiating is between the product agents, thus not disrupting the machine agents;
- They use a special infrastructure for the logistic subsystem, controlled by so called switch agents. Even though the practical implementation is akin to their solution, in our solution the service offered by the logistic subsystems can be considered as production steps offered by an equiplet and should be based on a more flexible transport mechanism.

However, there are important differences to our approach. The solution presented by Bussmann and Jenning has the characteristics of a production pipeline and is very useful as such, however, it is not meant to be an agile multi-parallel production system as presented here.

Other authors focus on using agent technology as a solution to a specific problem in a production environment. In [9] a multi-agent monitoring system is presented. This work focusses on monitoring a manufacturing plant. The approach we use monitors the production of every single product. The work of Xiang and Lee [10] presents a multiagent-based scheduling solution using swarm intelligence. Their work uses negotiating between job-agents and machine-agents for equal distribution of tasks among machines. The implementation and a simulation of the performance is discussed. In our approach the negotiating is between product agents and load balancing is possible by encouraging product agents to use equiplets with a low load. We did not focus on a specific part of the production but we developed a complete production paradigm based on agent technology in combination with a production grid. This model is based on two types of agents and focuses on agile multiparallel production. There is a much stronger role of the product agent and a product log is produced per product. This product agent can also play an important role in the life-cycle of the product [11].

In agent-based manufacturing the term holon is often used. While agent technology emerged from the field of computer science, the concept holon has its origin in computer integrated manufacturing (CIM) [12]. The concept was proposed by Koestler [13]. Parts of a system can be autonomous and stable on their own, but by cooperation they may form a bigger whole. This bigger whole could again be a part of an even bigger whole. A holon is both a part and a whole. A holon can represent a physical object or a logical activity. In the domain of manufacturing this can be a production machine, a production order or a human operator [14]. Agent technology can be used to implement a holon. This is where the two approaches, agent technology and the holon concept, meet. An important difference is that a holon can also be a passive entity like the aforementioned production order, while agents represent active autonomous entities. Fisher [15] uses a holonic approach for manufacturing planning and control. His work is based on the use of the Integration of Reactive behaviour and Rational Planning (InterRRap) agent architecture proposed by Müller [16]. Agents represent the holonic manufacturing components, forming a multiagent system. In our manufacturing model the holonic approach was not used, because a more simple multiagent system fitted our requirements.

III. GRID MANUFACTURING

In grid production, manufacturing machines are placed in a grid topology. Every manufacturing machine offers one or more production steps and by combining a certain set of production steps, a product can be made. This means that when a product requires a given set of production steps and the grid has these steps available, the product can be made [2]. The definition of a production step as used in this article is:

Definition[Production step] *A production step is an action or group of coordinated or coherent actions on a product, to bring the product a step further to its final realisation. The state of the product before and after the step are stable, meaning that the time it takes to do the next step is irrelevant and that the product can be transported or temporarily stored between two steps.*

The software infrastructure that has been used in our grid is agent-based. Agent technology opens the possibilities to let this grid operate and manufacture different kinds of products in parallel, provided that the required production steps are available [3]. The manufacturing machines that have been built in our research group are cheap and versatile. These machines are called equiplets and consist of a standardized frame and subsystem on which several different front-ends can be attached. The type of front-end specifies what production steps a certain equiplet can provide. This way every equiplet acts as a reconfigurable manufacturing system (RMS) [17]. An example of an equiplet front-end is a delta-robot. With this front-end, the equiplet is capable of pick and place actions. A computer vision system is part of the frontend. This way the equiplet can localise parts and check the final position they are put in. A picture of an equiplet with a delta-robot front-end is shown in Figure 1. For a product to be made a sequence of production steps has to be done. More complex products need a tree of sequences, where every sequence ends in a half-product or part, needed for the end product. The actual production starts at the branches of the tree and ends at the root. The equiplet is represented in software by a so-called equiplet agent. This agent advertises its capabilities as production steps on a blackboard that is available in a multiagent system where also the so-called product agents live. A product agent is responsible for the manufacturing of a single product and knows what to do, the equiplet agents knows how to do it. A product agent selects a set of equiplets based on the production steps it needs and tries to match these steps with the steps advertised by the equiplets. The planning and scheduling of a product is an atomic action, done by the product agent in cooperation with the equiplet agent and takes

Figure 1: An equiplet with a delta-robot frontend.

seven stages [4]. First, Let us assume that a single sequence of steps is needed:

1) From the list of production steps, build a set of equiplets offering these steps;
2) Ask equiplets about the feasibility and duration of the steps;
3) Indicate situations where consecutive steps on the same equiplet are possible;
4) Generate at most four paths along equiplets;
5) Calculate the paths along these equiplets;
6) Schedule the shortest product path using first-fit (take the first opportunity in time for a production step) and a scheduling scheme known as earliest deadline first (EDF) [4];
7) If the schedule fails, try the next shortest path.

For more complex products, consisting of a tree of sequences, the product agent spawns child agents, which are each responsible for a sequence. The parent agent is in control of its children and acts as a supervisor. It is also responsible for the last single sequence of the product. In Figure 2, the first two halfproducts are made using step sequences $< \sigma_1, \sigma_2 >$ and $< \sigma_3, \sigma_4 >$. These sequences are taken care of by child agents, while the parent agent will complete the product by performing the step sequence $< \sigma_4, \sigma_7, \sigma_2, \sigma_1 >$.

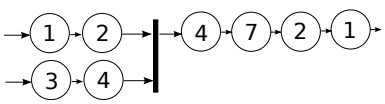


Figure 2: Manufacturing of a product consisting of two half-products.

Every product agent is responsible for only one product to be made. The requests for products arrive at random. In the implementation we have made, a webinterface helps the end-user to design or specify his or her specific product. At the moment, all features are selected a product agent will be created. During manufacturing a product is guided by the product agent from equiplet to equiplet. In Figure 3,

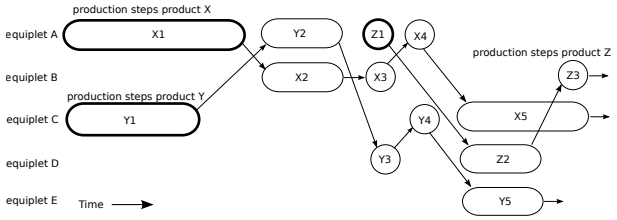


Figure 3: Three products in production.

the situation is shown for three products X, Y and Z using five different equiplets (A, B, C, D, E). Production step i of product X is denoted by Xi. From Figure 3 the following properties become clear:

- Production steps can differ in length as opposed to batch processing, where every step in the production line should normally take the same amount of time;
- Production can start at random;
- It will be unlikely that all equiplets are used at 100%;
- A failing production step on a product will not block the whole manufacturing process of other products as in a production pipeline used in mass production;
- Products have their own unique paths along the equiplets.

The path the product has to follow during manufacturing will in general be a random walk along the equiplets. Figure 4 gives an impression of such a random walk.

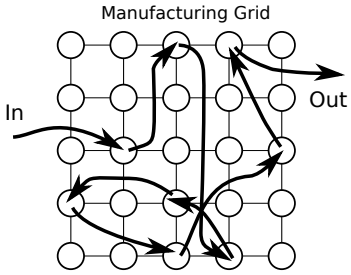


Figure 4: Random walk of a product in the grid.

This random walk is more efficient when the equiplets are in a grid arrangement against a line arrangement as used in batch processing. Some calculations on the average number of hops has been done for a random path between nodes on a line, on a circle and in a grid. In Figure 5, the number of hops is plotted against $\sqrt{N}$, where N is the number of nodes among the line, the circle or in the grid. The increase of the average path length (number of hops) is the highest for nodes put on a line. So a random walk along a line is behaving bad, when the number of nodes increases.

Figure 6 shows the global system architecture. The multiagent system is a distributed system consisting of computers belonging to the equiplet hardware where the equilet agents (EqA) live and some general computer platforms. The general computer platforms contain the product agents (PA) and blackboards that are used for sharing information that

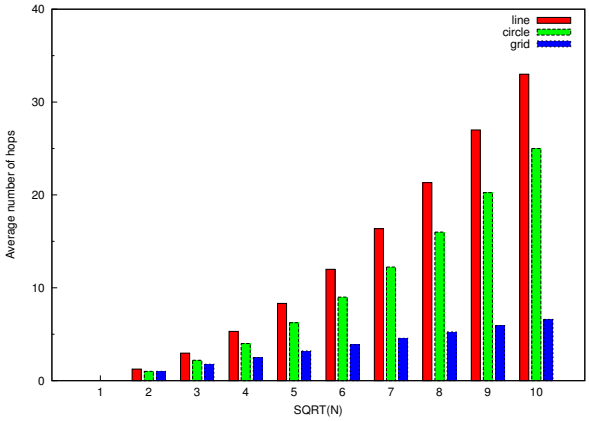


Figure 5: Number of hops for different configurations of N nodes.

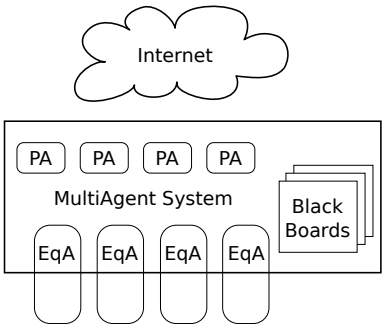


Figure 6: Global architecture of the software infrastructure.

should be available for all agents. The platforms are connected using standard Ethernet. New request for products to be built are received from the internet. Such a request will spawn a new product agent, that will plan its production path and schedule its production using the information available on the blackboards as described earlier in this section.

IV. Transport in the Grid

In the production grid, there is at least the stream of products to be made. Another stream might be the stream of raw material, components or half-products used as components. We will refer to this stream as the stream of components. These components could be stored inside the equiplets, but in that case there is still a stream of supply needed in case the locally stored components run short. This increases the logistic complexity of the grid model. In the next subsections, models will be introduced that alleviate the complexity by combining the stream of products with the stream of components.

A. Building box model

In the building box model, a tray is loaded with all the components to create the product. To maintain agility, this set of components can be different for every single product. Before entering the grid, the tray is filled by passing through a pipeline with devices providing the components. In this phase a building box is created that will be used by the grid to assemble the product. The equiplets in the grid are only used for assembling purposes. Figure 7 shows the setup.

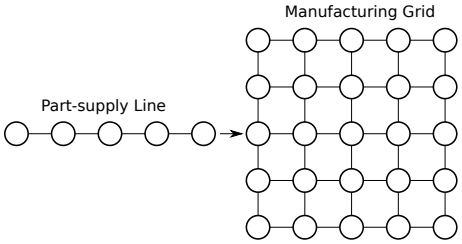


Figure 7: Production system with supply pipeline.

A problem with the previous setup is the fact that more complex products should be built by combining subparts that should be constructed first. In the previously presented setup all parts needed for the construction of the subparts should be collected in the building box, making the assembling process more complicated. Another disadvantage of putting all components for all subparts together in a building box is that this slows down the production time, because normally subparts can be made in parallel. A solution is shown in the setup of Figure 8. Subparts can be made in parallel and are input to the supply-line that eventually could be combined with the original supply-line.

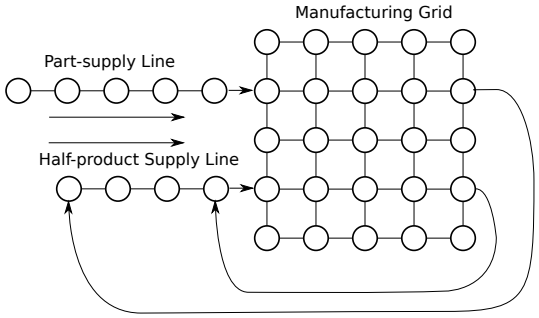


Figure 8: Production system with loops.

The next refinement of the system is presented in Figure 9. Here a set of special testnodes has been added to the system. These nodes are actually also equiplets, but these equiplets have a front-end that makes them suited for testing and inspecting final products as well as subparts that should be used for more complicated products.

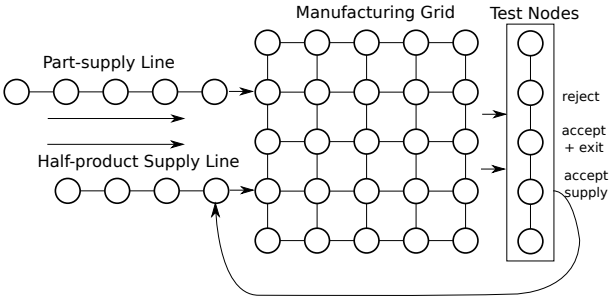


Figure 9: Production system with tests and loops.

A test can also result in a reject and this will also inform the product agent about the failure. If the product agent is a child

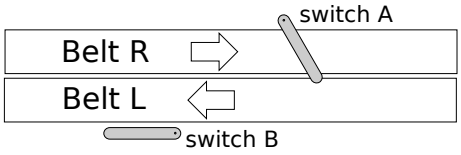


Figure 10: Bidirectional conveyor belt with switches.

agent constructing a subpart, it should consult the parent agent if a retry should be done. In case it is the product agent for the final product, it should ask its maker what to do.

B. Conveyor belt-based systems

A conveyor belt is a common device to transport material. Several types are in use in the industry. Without going into detail, some kind of classification will be presented here:

- belts for continuous transport in one direction;
- belts with stepwise transport from station to station. These types of belts can be used in batch environments, where every step takes the same amount of time and the object should be at rest when a production step is executed;
- belts with transport is two directions. This can also be realised by using two one direction belts, working in opposite direction.

In Bussmann [7], an agent-based production system is built using transport belts in two directions where a switch mechanism can move a product from one belt to another. A special switch-agent is controlling the switches and thus controlling the flow of a product along the production machines. In Figure 10 this solution is shown. Switch A is activated and will shift products from belt R to the belt L that will move it to the left. This concept fits well in the system developed by Bussmann, because that system is actually a batch-oriented system. In a grid the use of conveyor belts might be considered, but for agile transport several problems arise, giving rise to complicated solutions:

- should the direction in the grid consist of one-way paths or should be chosen for bidirectional transport?
- a product should be removed from the moving belt during the execution of a production step. A stepwise transport is inadequate, because of the fact that production steps can have different execution times in our agile model. This removal could be done by a switch mechanism as used by Bussmann, but every equiplet should also have it own switch-unit to move the product back to the belt.
- because the grid does not have a line structure for reasons explained in the first part of this paper, a lot of crossings should be implemented. These crossings can also be realised with conveyor belt techniques, but it will make the transport system as a whole expensive and perhaps more error-prone.

C. Transport by automatic guided vehicles

An alternative for conveyor belts is the use of automatic guided vehicles (AGV). An AGV is a mobile robot that follows certain given routes on the floor or uses vision, ultrasonic sonar or lasers to navigate. These AGVs are already used in industry mostly for transport, but they are also used as moving assembly platforms. This last application is just what is needed in the agile manufacturing grid. The AGV solution used to be expensive compared to conveyor belts but some remarks should be made about that:

- The AGVs offer a very flexible way for transport that fits better in non-pipeline situations;
- Low cost AGV platforms are now available;
- From the product agent view, an AGV is like an equiplet, offering the possibility to move from A to B.
- A conveyor-belt solution that fits the requirements needed in grid productions will turn out to be a complicated and expensive system due to the requirements for flexible transport.

In the grid, a set of these AGVs will transport the product between equiplets and will be directed to the next destination by product agents.

1) AGV system components: An AGV itself is a driverless mobile robot platform or vehicle. This AGV is mostly a battery-powered system. To use an AGV, a travel path should be available. When more than one AGV is used on the travel path, a control system should manage the traffic and prevent collisions between the AGVs or prevent deadlock situations. The control system can be centralised or decentralised.

2) AGV navigation: There are plenty ways in which navigation of AGVs has been implemented. The first division in techniques can be made, based on whether the travel path itself is specially prepared to be used by AGVs. This can be done by:

- putting wires in the path the AGV can sense and follow;
- using magnetic tape along the path to guide the AGV;
- using coloured paths, by using adhesive tape on the path to direct the AGV;
- using transponders, so the AGV can localise itself.

The second type of AGV does not require a specially prepared path. In that case navigation is done by using:

- laser range-finders
- ultrasonic distance sensors
- vision systems

Though it might look as if the decision for using AGVs has already been made, further research should be done to see what the efficiency will be for several implementations. This will be the subject of the next two sections.

V. SOFTWARE TOOLS

Two simulation software packages have been built. A simulation of the scheduling for production and a simulation for the path planning. The path planning tool will be used to calculate the efficiency for different transport interconnections. The scheduling tool will be used to calculate the number of active product agents within the grid. This number is important, because it will tell how many products should be temporally stored, waiting for the next production step to be executed.

A. Path planning simulation software

A path planning tool has been built, to calculate a path a certain product has to follow along the equiplets. The Dijkstra path algorithm has been used [18]. The tool can work on different grid transport patterns. This tool will be used to study several possible grid topologies. A screen-shot of the graphical user interface of the tool is shown in Figure 11. Several different topologies and interconnections can be chosen by clicking the appropriate fields in the GUI. The average transport path for all nodes is one of the results of this simulation.

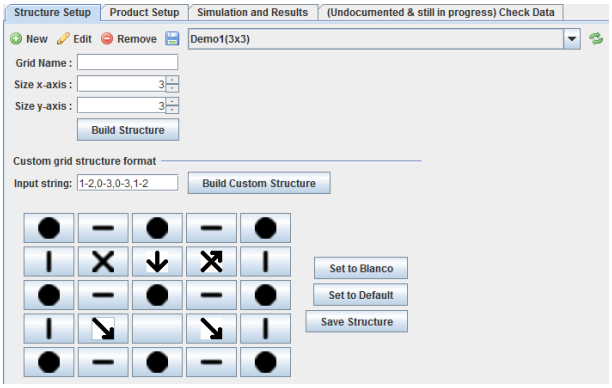


Figure 11: Path planning GUI.

B. Scheduling simulation software

The software for the scheduling simulations consists of two parts. One part is a command-line tool that is driven by a production scenario of a collection of product agents, each having their own release time, deadline and set of production steps. This production scenario is a human readable XML-file. The second part is a GUI for visualisation of the scheduling system. In Figure 12, a screen shot of this visualisation tool is shown.

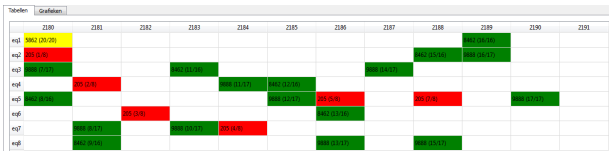


Figure 12: GUI of the scheduling simulator.

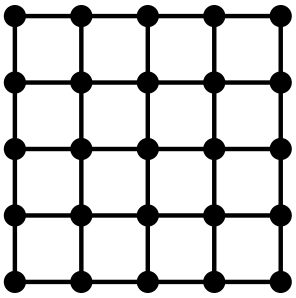


Figure 13: Standard fully connected grid.

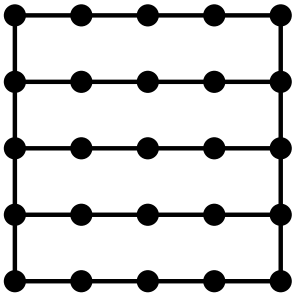


Figure 14: Grid with bidirectional lanes and bidirectional backbone lanes.

VI. RESULTS

This section shows two types of results. First, the transport possibilities are investigated using the path planning tool. Next, the results of the scheduling simulations are discussed.

A. Transport possibilities

To calculate the average pathlength in the grid for different paths, several structures have been investigated. Some of these structures were chosen to fit conveyor belt solutions of some type. All structures will also fit within the AGV-based solution.

1) A fully connected grid. where all paths are bidirectional paths as in Figure 13.
2) A grid where all paths are bidirectional, but this design has removed the crossings as in Figure 14. This structure could be implemented by conveyor belts in combination with switches;
3) A structure with five unidirectional paths and two bidirectional paths as in Figure 15. This structure is also a possible implementation with conveyor belts;
4) A structure with bidirectional paths combined in a single backbone as in Figure 16;
5) A structure with five bidirectional paths and two unidirectional paths as in Figure 17;
6) A fully connected grid, but now with half of the paths unidirectional as in Figure 18.

For all these structures the average path is the result from a simulation of 1000 product agents, all having a random walk within the grid. Each product agent has an also random set of equiplets it has to visit ranging from 2 to 50 equiplets per product agent. Every path or hop between adjacent nodes is considered to be one unit length. If the paths have no crossings, a conveyor belt might be used, because crossing belts will

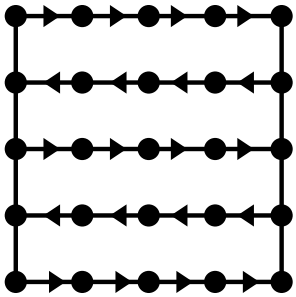


Figure 15: Grid with unidirectional lanes and bidirectional backbone lanes.

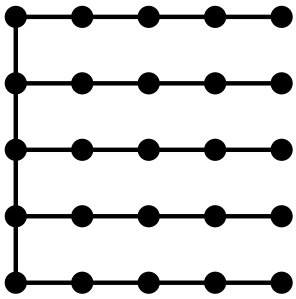


Figure 16: E-shaped connection, with bidirectional lanes.

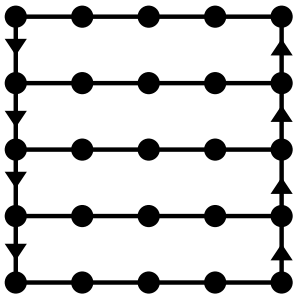


Figure 17: Bidirectional lanes with unidirectional backbones.

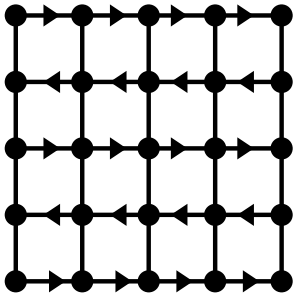


Figure 18: Fully connected grid with unidirectional lanes.

result in a more complex system. All structures can also be implemented with AGVs. For some structures the average path can also easily be calculated and the results of these exact calculations are comparable with the simulation results.

The results of the simulation are given in a table and also plotted as a histogram in Figure 19. In Table I, a second outcome from the simulation is also shown. This is the percentage of agents that could find an alternative path of

TABLE I: Results of the simulation.

Structure	1	2	3	4	5	6
Average path	3.2	3.9	6.4	5.1	6.0	3.6
% Alternatives	60	16.7	8.4	0	0	27

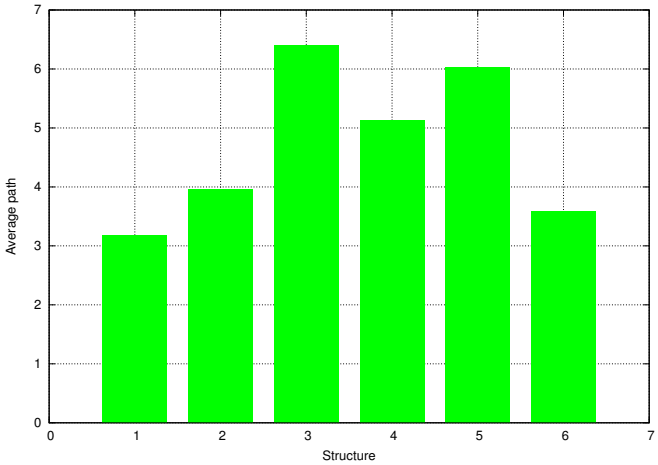


Figure 19: Simulation results for different structures.

the same length. This result is of interest when in a traffic control implementation, alternative paths become important.

As could be expected, the best result is achieved in the fully connected grid with bidirectional paths. Changing the grid to an almost identical structure of Figure 18 with unidirectional paths, results in only a small penalty. This structure could also be useful in an AGV-based transport system, reducing collision problems because of the one-way paths used. Both structures also offer a relative high percentage of alternative paths, that could also be useful in an AGV-based system. The structures that fit a conveyor belt solution show a path length that is considerably higher.

When AGVs are used, the architecture of the production system slightly changes. The solution fits well in the software infrastructure. For the product agent, the AGV can be seen as just another equiplet, but instead of providing production steps, transport in the grid is offered. Another difference is that the product agent will be tied to an AGV during the whole production. The resulting architecture is shown in Figure 20. The AVGs are represented by transport agents (TA). During the execution of a production step, the equiplet agent can also cooperate with the transport agent to put the product on the AGV in the right position. This might come handy when a product is too large for the equiplet to handle by itself.

B. Scheduling

The next results were generated using the scheduling tool. This tool was used in earlier research [4] to discover the scheduling approach to be used. The scheduling is based on timeslots, having a certain duration. Such a timeslot is the minimal allocatable unit of time. The methods used for scheduling were derived from real time scheduling schemes adapted to the multiagent environment. To explain these schemes, some symbols used in expressions should be defined:

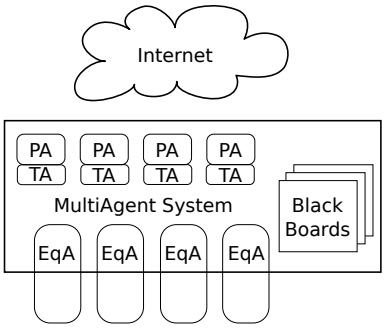


Figure 20: Global architecture with transport.

P is the product set. A single product is denoted as P_i.

r_i is the first timeslot after release of product P_i

d_i is the timeslot for the deadline of product P_i

τ is the current timeslot

$s_i(\tau)$ is the number of timeslots of product P_i that is left to be done.

Five well-known scheduling schemes are as follows.

1) Fixed priority, FP. Every task is assigned a priority depending on the task type. The highest priority tasks are completed before the lower priorities are run.
2) Earliest deadline first, EDF. The task with the first deadline to come gets the highest priority and is handled first.
3) Least slack first, LSF. The task with the minimum slack gets the highest priority and is handled first. Slack is defined as the total time available until the deadline minus the time to complete the task. The slack for product P_i at timeslot τ can be written as $(d_i - \tau) - s_i(\tau)$.
4) Smallest critical ratio first, CR. The critical ratio is defined as the total number of timeslots available divided by the number needed. For a product P_i at timeslot τ: $(d_i - \tau)/s_i(\tau)$. If this number turns out to be 1, all timeslots should be used. If it is lower than 1, the scheduling is infeasible. A high number shows that many slots are available for a relative small number of needed timeslots.
5) Shortest process first, SPF. The task with the shortest time to complete gets the highest priority.

All these types can be used in conjunction with what is called preemption. By this is meant that when a higher priority task arrives, another already running task is paused (preempted) to make way for the higher priority task. After completion of the higher priority task, the preempted task is resumed. Because all agents are equal, fixed priority is inadequate as a scheduling scheme for the production grid. Both EDF and LSF are considered optimal in the sense of feasibility: this means that if there exists a feasible schedule, EDF or LSF will find it [19]. However, this is only true for the situation where a single resource is scheduled among requesters, as is the case in a single processor computer system, where the processor time is scheduled among different tasks.

The adjustments that has been made to adapt the scheduling schemes to the situation of the production grid had to do with preemption and the way a feasible scheduling and a failing scheduling are treated. For scheduling in the grid the following list of objectives has been worked out:

1) It should offer a best effort to schedule products that will arrive at random times.
2) It should schedule products at high grid loads.
3) It should be fast and reliable. The scheduling should take a small amount of time.
4) It should introduce only a small intercommunication overhead. This will mean that the amount of inter-agent messages should be kept low.
5) It should be fair. When a product is scheduled for production with a feasible scheduling, meaning the product will be completed before the deadline, the scheduling system should be designed such that it will guarantee that the feasible scheduling will not be changed to an infeasible scheduling at a later time by the scheduling system.

In [4], two approaches for preemption are introduced. To describe the differences EDF will be used as an example. If a product arrives at time t with deadline T_d, two scenarios are possible:

1) All products with a later deadline will temporarily give up their schedules starting from t to make way for the newly arrived product. This product will be scheduled and the products that gave up their scheduling try to reschedule within the constraints for their deadlines according to the EDF scheme. However, if one of these reschedules fails, objective 5 of the scheduling system is not achieved. This will result in reporting a scheduling failure for the newly arrived product and a restore of the schedules of the already active products. This approach is called the strong approach.
2) The newly arrived product will first try to schedule its production without disturbing the other products. Only if it fails it will follow the schedule and reschedule approach mentioned in the first scenario. This approach is called the weak approach.

The implementation of these two approaches is done by sending broadcast messages as well as agent to agent messages. In case of an infeasible schedule for the newly arrived agent, this agent will broadcast its deadline to all active product agents. In reply to this broadcast, agents with a later deadline will send their claimed production steps to the new agent and this will try to schedule its production assuming these claimed steps are now available. If the scheduling succeeds it will try to reschedule all other active agents. In case of success it will adjust the scheduling information on the blackboard involved with the new scheduling and send the new schedules to the participating agents. By locking the access to the blackboard by other agents during (re)scheduling, this scheduling action is atomic.

In [4], it is shown that the approaches weak and strong result in almost the same rate of successful schedules, having an average difference in results below 0.5%. Both approaches

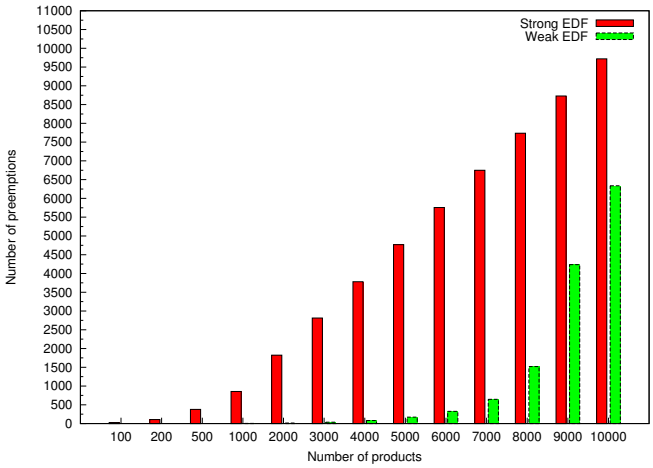


Figure 21: Number of preemptions for strong and weak EDF.

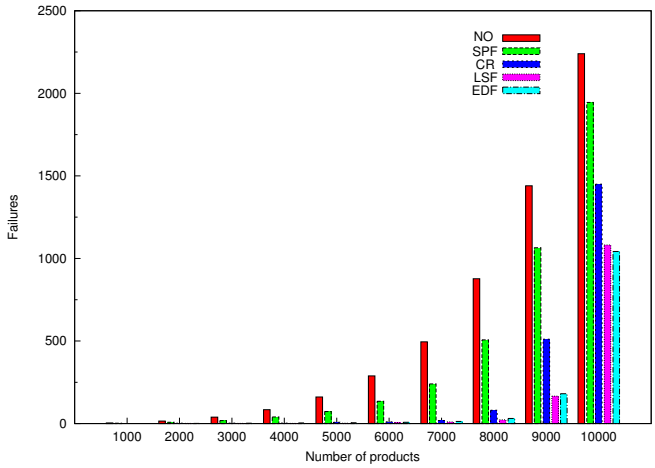


Figure 23: Failure-count for different scheduling algorithms.

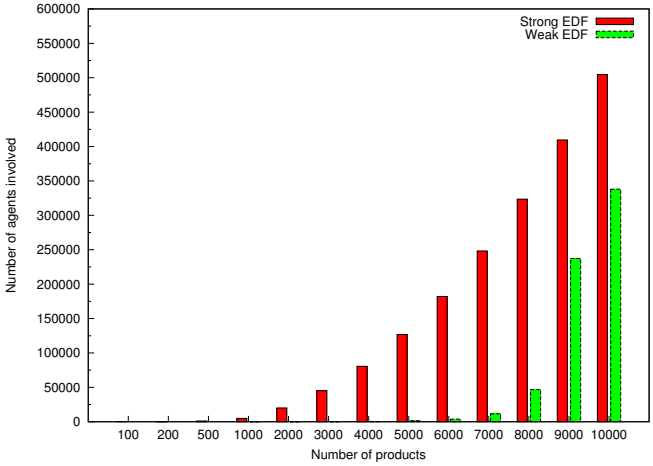


Figure 22: Number of agents involved for strong and weak EDF.

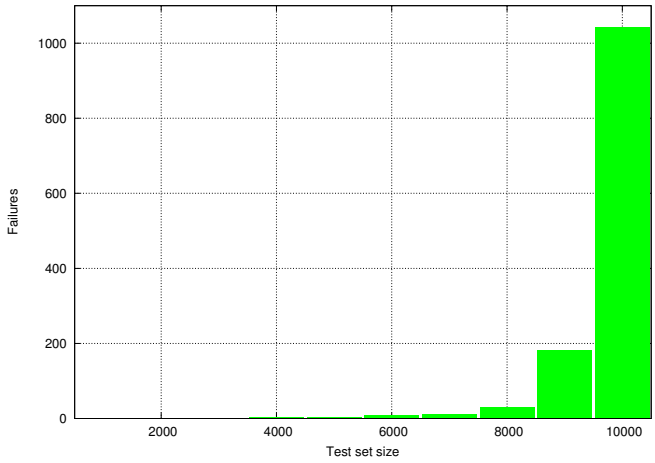


Figure 24: Simulation Results for different sizes op product sets.

fulfill objectives 1 and 2. In Figure 21, the number of preemptions for both strong and weak versions of EDF are plotted for different sizes of test sets. Another important value that gives and impression of the inter-agent communication overhead is shown in Figure 22. From both figures it becomes clear that the amount of overhead in inter-agent communication and rescheduling calculations used by the strong version is much higher than the weak version. Objectives 3 and 4 are more feasible using the weak version. Both versions were already by design compliant with objective 5. When we consider the different scheduling algorithms mentioned before, leaving out fixed priority, and using the weak approach the resulting number of failures for different numbers of products are plotted in Figure 23. Earliest deadline first (EDF) turned out to be a good choice. The success rate is comparable to LSF, but the advantage of EDF over LSF is that the deadline is a constant value while the slack changes over time and has to be recalculated, introducing an extra small overhead.

For the previous and coming simulations, the following conditions were used:

- Every product agent has a random number of equiplets to visit ranging from 1 to 20 with an average of 10 equiplets.
- The number of timesteps in a simulation run is 10000 and the duration of a production step is 1 timestep.
- The time window between release time and deadline of a product is random between 1 to 20 times the total production time of a product. This total production time in timesteps is in this case equal to to number of equiplets the product agent has to visit.
- Each equiplet is offering a single unique production step.

For a set with 10000 product agents, each having an average number of 10 production steps, the amount of production steps needed is 100000. During 10000 timesteps the grid consisting of 10 equiplets is actually offering a maximum of 100000 production steps. Figure 24 shows that in the given situation of 10000 product agents the number of scheduling failures is over 1000. In Figure 25, the average number of product agents in a manufacturing grid consisting of 10 equiplets is shown for different sizes of test sets. This information shows

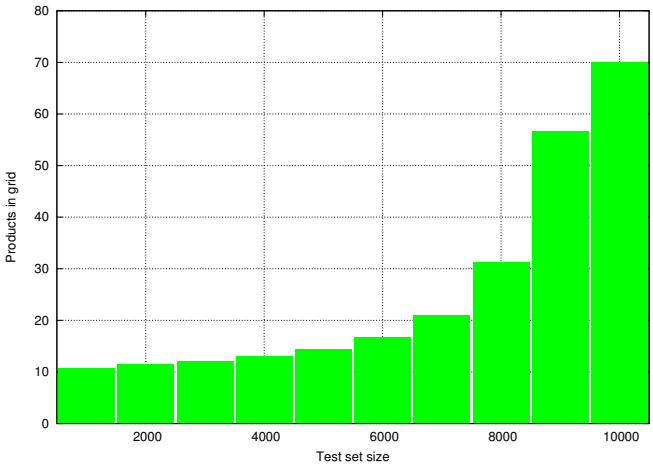


Figure 25: Simulation Results for different sizes op product sets.

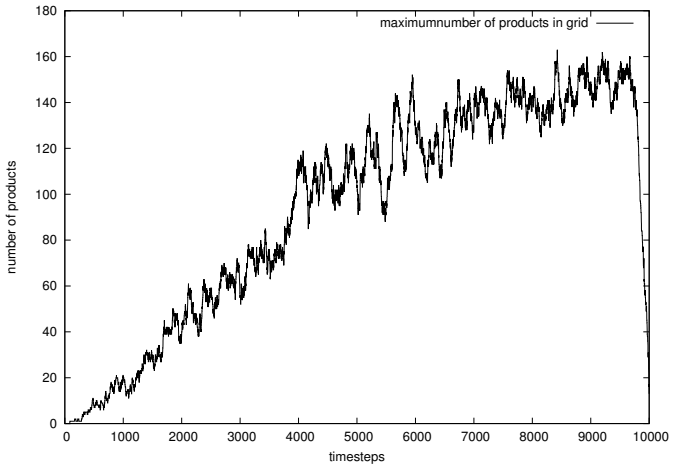


Figure 27: Maximum number of active products in the grid.

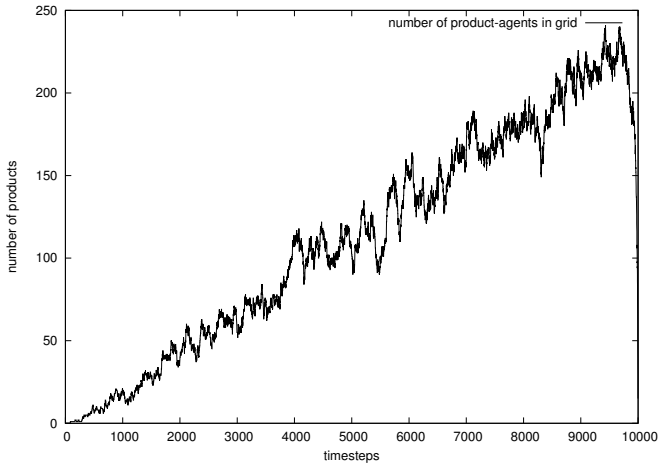


Figure 26: Increasing number of active products in the grid.

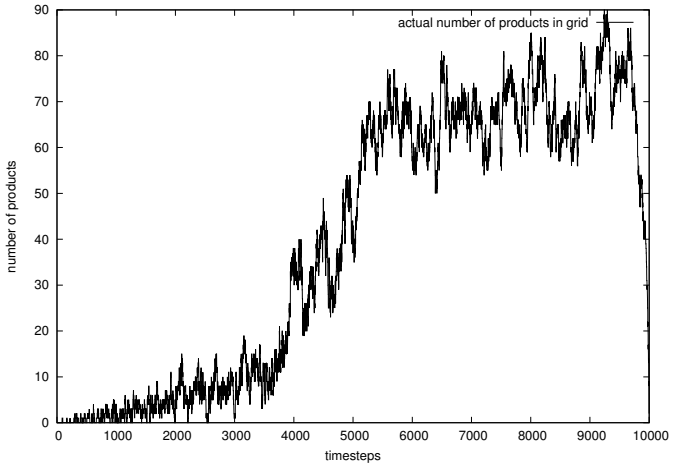


Figure 28: Actual number of products in the grid.

that the number of products exceeds the number of equiplets by far above 8000 products. Given the aforementioned test conditions, a number of 8000 products compares to a load of 80%. At a load of 80% the number of products, which are not handled by equiplets and have to wait, is two times the number of equiplets, resulting in an average of two AGVs waiting for service by an equiplet.

Another simulation has been set up to study the behaviour of the grid under increasing load. This simulation is based on a scenario with a linear increasing amount of product agents in time as shown in Figure 26. In this situation, a product is considered active in the grid between its release time and its deadline. Because of the fact that at the end of the graph, the grid is actually overloaded, the maximum number of active agents in the grid will not increase due to the fact that more and more products will have an infeasible scheduling and will not contribute to the number of products in the grid. This effect is shown in Figure 27. The actual number of products in the grid is shown in Figure 28. This graph is not based on release time and deadline, but on release time and time of completion. When we look at the actual number of active products in the grid, the resulting graph shows an remarkable shape. In the beginning, the actual number is even less than the number plotted in the graph of Figure 26. This is due to the fact that in a grid that is only used by a small amount of product agents, every product will be finished far before its deadline. A finished product is now not considered active in the grid any more. However, at a certain point there is a steep increase in the number of products and the graph saturates at the same level of 70 products as shown in Figure 25 for a test set of 10000. The number of rejected products due to a failing scheduling will increase. This also means that overloading the grid will generate many active products that should be stored somewhere, because in this given situation, only 10 products can be handled by an equiplet.

VII. Conclusion

For the agile grid-based and agent-based manufacturing the buildingbox as well as the AGV-based system offer advantages:

- By using a building box, the transport of parts to the assembling machines (equiplets) is combined with the transport of the product to be made. It will not happen that a part is not available during manufacturing;

- Because the product as well as it parts use one particular AGV during the production, there is never a competition for AGV during the manufacturing process;
- An AGV can use the full possibility and advantage of the grid-based system being a compact design resulting in short average paths;
- The product agent knows which equiplets it should visit and thus can use the AGV in the same way as an equiplet. The product agent can instruct the AGV agent to bring it to the next equiplet in the same way as it can instruct an equiplet agent to perform a production step;
- An AGV can bring the production platform exact to the right position for the equiplet and can even add extra movement in the X-Y plane or make a rotation around the Z-axis;
- If an AGV fails during production the problem can be isolated and other AGVs can continue to work. In a conveyor belt system a failing conveyor might block the whole production process.

There are also some disadvantages:

- There should be a provision for charging the battery of the AGV.
- Simulations show that the amount of agents in the grid shows a strong increase in a grid that is loaded over 80%. This will result in a lot of AGVs in the grid leading to traffic jam;
- Only products that fit within the building box manufacturing model can be made.

Agent-based grid manufacturing is a feasible solution for agile manufacturing. Some important aspects of this manufacturing paradigm have been discussed here. Transport can be AGV-based provided that the load of the grid should be kept under 80% to overcome the temporary storage requirements and problems with the communication and rescheduling overhead.

ACKNOWLEDGEMENT

The authors would like to thank Mathijs Kuijl, Bas Alblas, Jaap Koelewijn, Pascal Muller, Lars Stolwijk, Roy de Kok and Martijn Beek for their effort and contributions in developing the software tools.

REFERENCES

[1] L. v. Moergestel, E. Puik, D. Telgen, M. Kuijl, B. Alblas, J. Koelewijn, and J.-J. Meyer, "A simulation model for transport in a grid-based manufacturing system," Proceedings of the Third International Conference on Intelligent Systems and Applications (INTELLI 2014), Sevilla, Spain, 2014, pp. 1–7.

[2] E. Puik and L. v. Moergestel, "Agile multi-parallel micro manufacturing using a grid of equiplets," Proceedings of the International Precision Assembly Seminar (IPAS 2010), Chamonix, France, 2010, pp. 271–282. Springer ISBN-13: 978-3642115974.

[3] L. v. Moergestel, J.-J. Meyer, E. Puik, and D. Telgen, "Decentralized autonomous-agent-based infrastructure for agile multiparallel manufacturing," Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2011) Kobe, Japan, 2011, pp. 281–288.

[4] L. v. Moergestel, J.-J. Meyer, E. Puik, and D. Telgen, "Production scheduling in an agile agent-based production grid," Proceedings of the Intelligent Agent Technology (IAT 2012), Macau, 2012, pp. 293–298.

[5] M. Paolucci and R. Sacile, Agent-based manufacturing and control systems: new agile manufacturing solutions for achieving peak performance. Boca Raton, Fla.: CRC Press, ISBN-13: 978-1574443363, 2005.

[6] E. Montaldo, R. Sacile, M. Coccoli, M. Paolucci, and A. Boccalatte, "Agent-based enhanced workflow in manufacturing information systems: the makeit approach," J. Computing Inf. Technol., vol. 10, no. 4, 2002, pp. 303–316.

[7] S. Bussmann, N. Jennings, and M. Wooldridge, Multiagent Systems for Manufacturing Control. Berlin Heidelberg: Springer-Verlag, ISBN-13: 978-3642058905, 2004.

[8] N. Jennings and S. Bussmann, "Agent-based control system," IEEE Control Systems Magazine, vol. 23, no. 3, 2003, pp. 61–74.

[9] D. Ouelhadj, C. Hanachi, and B. Bouzouia, "Multi-agent architecture for distributed monitoring in flexible manufacturing systems (fms)," Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2000), San Francisco, CA, USA, 2000, pp. 2416–2421.

[10] W. Xiang and H. Lee, "Ant colony intelligence in multi-agent dynamic manufacturing scheduling," Engineering Applications of Artificial Intelligence, vol. 16, no. 4, 2008, pp. 335–348.

[11] L. van Moergestel, E. Puik, D. Telgen, H. Folmer, M. Grünbauer, R. Proost, H. Veringa, and J.-J. Meyer, Enhancing Products by Embedding Agents: Adding an Agent to a Robot for Monitoring, Maintenance and Disaster Prevention, ser. Communications in Computer and Information Science, J. Filipe and A. Fred, Eds. Springer Berlin Heidelberg, 2014, vol. 449, ISBN-13: 978-3662444399.

[12] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," Journal on Engineering Applications of Artificial Intelligence, vol. 22, issue 7, pp. 979–991, Pergamon Press, Inc. Tarrytown, NY, USA, 2009.

[13] A. Koestler, The Ghost in the Machine. Arkana Books, London, 1969, Reprinted 1990, Penguin Books, ISBN-13: 978-0140191929.

[14] S. Bussmann and D. McFarlane, "Rationales for holonic manufacturing control," Proceedings of the second international workshop on intelligent manufacturing systems, 1999, pp. 177–184.

[15] K. Fisher, "Agent-based design of holonic manufacturing systems," Robotics and Autonomous Systems, vol. 27, no. 1-2, 1999, pp. 3–13.

[16] J. Müller, The design of intelligent agents: a layered approach. Springer, 1996, ISBN 978-3540620037.

[17] Z. M. Bi, S. Y. T. Lang, W. Shen, and L. Wang, "Reconfigurable manufacturing systems: the state of the art," International Journal of Production Research, vol. 46, no. 4, 2008, pp. 599–620.

[18] M. Sniedovich, "Dijkstras algorithm revisited: the dynamic programming connexion," Control and Cybernetics, vol. 35, no. 3, 2006, pp. 599–620.

[19] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, Scheduling in Real-Time Systems. Chichester, West Sussex: John Wiley and sons, 2002, ISBN-13: 978-0470847664.

Opportunistic Use of Cloud Computing for Smart Mobile Applications: Realizing Qo*-awareness with Dynamic Decision Networks in Execution

Nayyab Zia Naqvi, Davy Preuveneers, and Yolande Berbers
iMinds-DistriNet, KU Leuven, Belgium
Email:{nayyab.naqvi, davy.preuveneers, yolande.berbers}@cs.kuleuven.be

***Abstract*—Smart and context-aware Mobile Cloud Computing applications challenge the way Quality-of-Service and Quality-of-Context are maintained when operating with a federated mobile cloud environment. Many resource and performance trade-offs exist for the federated deployment of smart mobile applications. Consequently, finding the best strategy to deploy and configure intelligent applications in this federated environment is a non-trivial task. We analyse the challenges and requirements for the dynamic deployment of smart applications in such a setting and present a quality-aware federated framework for the development and deployment of smart mobile applications to use the cloud opportunistically. Our framework utilizes Qo*-awareness and dynamic adaptability to account for the uncertain conditions given the partially observable context. Experiments with our framework demonstrate the feasibility and the potential benefits to automate the deployment and configuration decisions in the presence of a changing environment and runtime variability.**

Keywords–dynamic deployment; Bayesian models; mobile cloud computing; smart applications; decision support.

I. INTRODUCTION

Mobility and context-awareness have multiplied the use of mobile devices in homes or offices, health and cities giving them a *Smart* label. Intelligent and smart applications are gaining popularity day-by-day due to the ease and the control they offer to the user. Continuous advancements in mobile technology are changing our habits and the ways we interact with these mobile applications [1]. This rapidly evolving mobile technology is giving a momentum to the smartness of these devices and a better control of our mobile applications, considering the perspective of the user and his situation [2]. A wide range of sensing, communication, storage and computational resources makes these mobile devices a perfect platform for ubiquitous computing offering pervasive connectivity and a source for context-awareness.

Context-awareness is a congenital characteristic [3] of intelligent applications to the notion of adaptability and smartness. Context [4] is defined as any information that can be used to characterize the situation of an entity, where an entity can be an object, a place or a person relevant to the current scope of the system. Research and development of context-awareness is narrowing the gap between us, our devices and the environment. Context aims to become the fabric of ubiquitous computing. The consequent smart applications behave as the constituents of ubiquitous environments as envisioned by Mark Weiser [5].

The user takes a passive role in context-aware applications and these mobile applications decide on his behalf. These applications require continuous processing and high-rate sensors' data to capture the user's context, such as his whereabouts and ongoing activities as well as the runtime execution context on the mobile device. However, being smart requires being right in an adaptation decision. Context is dynamic in nature and is prone to ambiguity. This uncertainty not only leads to incorrect decisions but also makes proactive decision making impossible, impacting the *Quality-of-Service (QoS)* of the application in a negative fashion. We need certain attributes that signify the adequacy or degree of suitability of the context data. This degree of suitability, often regarded as the *Quality-of-Context (QoC)* [6], can highly affect the adaptation decision.

Despite the continuous improvements in mobile technology, the exponential growth in mobile usage is formidable to cope with the resource limitations. Mobile Cloud Computing (MCC) offers Mobile Cloud Augmentation (MCA) with the aim to empower mobile devices to run more demanding or long running tasks by providing plentiful storage and processing capabilities on cloud servers. Cloud computing [7] offers access to an always connected, decentralized, and abstracted infrastructure of a heterogeneous pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can scale up or down instantly in a controlled fashion. This federated design can be applied to almost any application and has been shown to improve both the speed and energy consumption [8]. Most of the cloud applications follow a thin client philosophy where resource intensive tasks are outsourced to the cloud infrastructure in a brute force fashion.

A modular design philosophy for smart applications is ideal enabling an optimal deployment in MCA. However, attaining the most suitable deployment and configuration strategy for these applications is not always clear and straightforward. The MCA federation for such type of applications is a non-trivial task. On the one hand, we have mobile devices with built-in sensors to sense the context of user. On the other hand, cloud resources can be utilized opportunistically to save resources for mobile users [9][10]. At the core of such a distributed environment, a decision of *what* to run *where* and *when* is gruelling under a changing runtime execution context. Figure 1 shows an overview of the federation in MCA for context-aware applications. There can be multiple deployment strategies for a smart application both at design time and at runtime. However, it is not clear which component should be deployed where, as the context changes at runtime. The crux of the problem lies in the decision of *when* to change the location of a component and decide where to deploy it in an adaptive manner. The Topology and Orchestration Standard for Cloud Applications (TOSCA) [11] is an xml-based standard to define the topology of cloud web services, their components, relationships, and the processes that manage them. It can also be used to define the orchestration of a mobile cloud application with an additional decision making

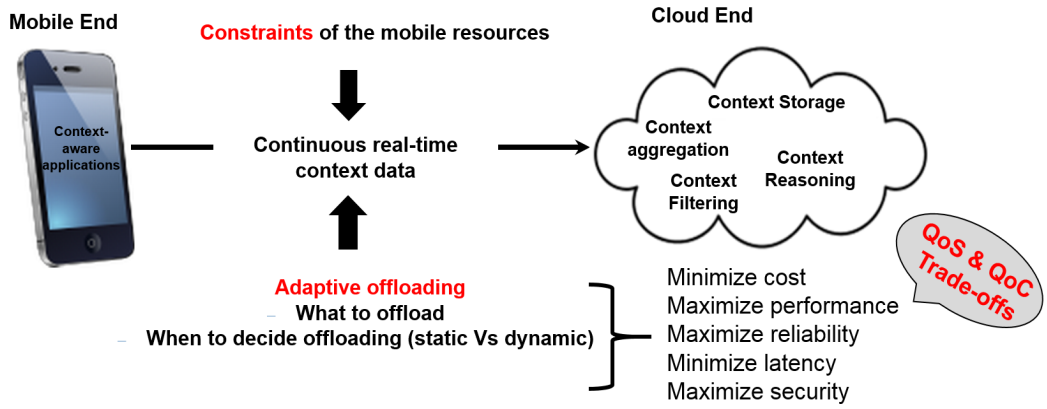


Figure 1. Overview of context-driven optimized federation in MCA.

support to choose the location of a component, based on the involved context factors and the resource-performance trade-offs. Abolfazli et al. [12] define decision making factors for the augmentation in MCC. The trade-offs between computations versus communication and performance versus latency are generally the most significant. User preferences with respect to privacy and security for sensitive data also play a role in the decision of which information to process where.

Computation on mobile devices always involves compromises and trade-offs [13] with respect to the required QoS [14] and QoC [6]. Context sensing and simple processing run on the mobile device and other computationally more expensive context management tasks like pattern detection, reasoning and learning are offloaded on the cloud infrastructure. The question is how we can effectively blend them to always achieve a feasible and beneficial augmentation? The constraints of a mobile environment play a role in the deployment decision. Additionally, The requirements of semantic knowledge and the resource characteristics of the application components make this decision highly dynamic. A primary challenge is to liberate device resources without compromising the QoS while preserving the QoC necessitating a Qo*-awareness in deployment decision making. Moreover, runtime uncertainty and erratic nature of the context information [15] periodically impact these deployment decisions and consequently performance; hence, the decision support needs to be flexible enough to ascertain the quality of its own decisions. The system should be aware of the impact of its decisions over time to optimize the runtime deployment and learn from its mistakes as humans do.

To address the needs of real-time dynamic systems, the Monitor-Analyze-Plan-Execute (MAPE-K) [16] framework with a concept of a feedback loop seems promising. We present a framework for self-adaptation of smart mobile applications as an instance of MAPE-K framework. Dynamic Decision Networks (DDNs), a specialized form of Bayesian Networks (BNs) are employed to analyze, plan, execute and update the knowledge-base at runtime with the changing context of the execution environment. Self-reinforcement [17] with the help of utility functions and temporal delays enables the framework to learn from its mistakes [18] in order to ascertain the quality of the taken adaptive decisions. Building on recent work of Bencomo et al. [19] in self-adaptive systems, we have adopted a model@Runtime approach applying DDNs, to incorporate the impact of trade-offs and to take an optimal decision under variability. DDN modelling is an emerging research topic, and researchers are investigating its use in the area of self-adaptation for autonomous systems in several domains [20][21][22][23].

In this work, we extend our previous work [1] and present a multi-objective application model with a fair trade-off between the required objectives. We highlight the challenges to exploit the cloud infrastructure opportunistically, and investigate the requirements to achieve a federation in MCA for smart mobile applications. Our presented framework is able to learn the deployment trade-offs of intelligent applications on the fly and is capable of learning from earlier deployment or configuration mistakes to better adapt to the settings at hand in a Qo*-aware fashion. To verify the effectiveness of our framework, a feasibility analysis is conducted on *Smart Lens*, an Augmented Reality (AR) based use case application. Research is conducted with respect to the communication cost and resource utilization when extending an AR application with context-awareness and cloud computing.

Our research offers the following contributions to support effective smart application deployment in a Qo*-aware federated environment:

- An investigation of the trade-offs that arise for data-intensive context-aware smart applications in MCA
- A federated dynamic deployment with respect to Qo* trade-offs
- Use of specialized probabilistic models to automatically learn the overhead of deployment trade-offs and compromises

This paper is structured in seven sections. In Section II, we give an overview of the background and the related work in MCA and discuss the gaps in state-of-the-art for federated deployments and decision making under uncertainty. Section III details our use case scenario motivating the need for smart deployment decisions. This is followed by Section IV, where we highlight the requirements and objectives of Qo*-aware decisions. Section V provides a brief account of our federated framework and the details about our approach of learning the trade-offs for dynamic deployments using a probabilistic decision model to mitigate the influence of runtime uncertainty. Finally, after evaluating our approach applied on our use case scenario in Section VI, the paper concludes and offers a discussion of topics of interest for future work.

TABLE I. The three offloading frameworks compared in terms of trade-off modeling and decision making.

Criteria	Features	MAUI	CloneCloud	ThinkAir	Our approach
Trade-Offs	Objective	Energy	Performance	Performance	Both
	Flexibility	Low	Medium	Medium	High
Decision	Context-Aware	High	Low	High	High
	Data store	Remote	Local	Local	Local
	Approach	Optimization	Optimization	Rules	Network
	Metric	Cost	Cost	Property	Property

II. Background and related work

In this section, we discuss and examine related work in the area of MCA, i.e., an active research domain in MCC. The strategies for runtime optimization in the presence of uncertain operational conditions are also briefed here.

A. Mobile cloud augmentation (MCA) and Qo*-awareness

Outsourcing the computation to the cloud can be beneficial to achieve an ideal QoS [14][24]. It has been proven to reduce the resource load on mobile devices [24], also for context-aware data-intensive applications [25]. This combination may result in improved energy efficiency and a reduced load on resources, such as CPU and memory. Beside context-awareness, a number of use cases combining data intensive mobile application and cloud computing have been described in the literature [26][27]. However, the use of context-awareness in cloud computing is often approached from a functional standpoint. The question for federated deployment is where do you draw the line? What should be run on the cloud and what work should be done by the mobile device? The easiest option is to have everything stay local and not use the cloud or Internet at all, but as mentioned earlier, this could lead to bad performance of the application. Since the cloud has enormous processing power, it is also possible to adopt a thin client approach and do almost all the work online. However, we cannot forget that communication with the cloud has a price as well, economically and in terms of time and energy. In this case, multiple conflicting objectives affect the decisions and the distribution is never clear-cut in the scenario of intelligent systems considering trade-offs with respect to QoS and QoC [13]. Furthermore, acquiring knowledge from the available data for context information, requires a trustworthy mechanism where the ambiguity and uncertainty in context data can be mitigated.

Previous research [28][29] was carried out on how to realize platforms that allow the applications to make the partitioning decision while it is running, providing the user with the best experience possible. Even more impressive, some of these platforms can let applications enjoy the best of computation offloading by only making minor changes [28]. Context-awareness and computation offloading is added to achieve the desired functionality but the accompanying trade-offs regarding deciding *what* to offload and *when* are not explored. Narayanan et al. [30] predict the resource consumption on the basis of historical data by applications. They use this data to modify the fidelity of an application, based on the inputs and parameters received by any mobile application at runtime. Huerta and Lee [31] discuss a profiling based smart offloading policy using historic resource usage data. However, processing entire history logs is cumbersome. Cuckoo [32], ThinkAir [29] and MAUI [33] present an MCA model based on multi-objective criteria with respect to the performance and energy consumption. Cuckoo offers static offloading decisions without context-awareness. ThinkAir and MAUI make a decision based on the execution time, energy consumption, and previous execution history. MAUI processes the offloading requests by using the historic data to predict the execution time of any task without considering the input size of that execution, resulting in wrong prediction and offloading decisions [12]. We show a comparison of our approach with closely related approaches in Table I.

Chen et al. [35] investigate challenges related to the fact that each platform has its own capabilities and limitations to achieve certain QoS requirements. They present a context-aware resource management approach for service oriented applications with the ability to handle the inherent service and network dynamics and to provide end-to-end QoS in a secure way. QoC attributes and their modelling comes into play to capture the uncertainties in context data. Bellavista et al. [36] discuss the QoC requirements and their impact on context usage. Sheikh et al. [37] identify several quality indicators like precision, freshness, spatial resolution, temporal resolution and probability of correctness. The authors propose that these quality indicators are well-suited in ubiquitous systems for healthcare. However, no quantification mechanism has been proposed by the authors in order to evaluate the role of these parameters in critical decision support. Kim et al. [38] present the quality dimensions such as accuracy, completeness, representation consistency, access security and up-to-dateness for measuring QoC in ubiquitous environments.

Our solution starts without historic information and uses only the context at hand to predict the Qo*-aware dynamic deployment scheme for each component of a smart mobile application. Our solution does not focus on the partitioning scheme, but the optimal decision making for each of the component cloned on both the mobile and cloud ends.

B. Decision support under changing circumstances

Restating the obvious, intelligent applications have to take into account the runtime uncertainty in context data. Context sources are dynamic in nature. They can disappear and re-appear at any time and context models change to include new context entities and types. The properties of context sources and context types can change randomly and the uncertainty can vary too. Pearl [39] explains the problem of uncertainty and argues that extensional or rule based systems cannot perform well under uncertainty. Probabilistic theory allows complex reasoning with a combination of observed evidences. Probabilistic systems can handle unseen situations addressing the influence of involved uncertainty. In our work, we are concerned with the mechanism of deciding when to reconfigure the deployment under a changing context at runtime.

Context-aware applications borrow decision models from artificial intelligence and machine learning field such as su-

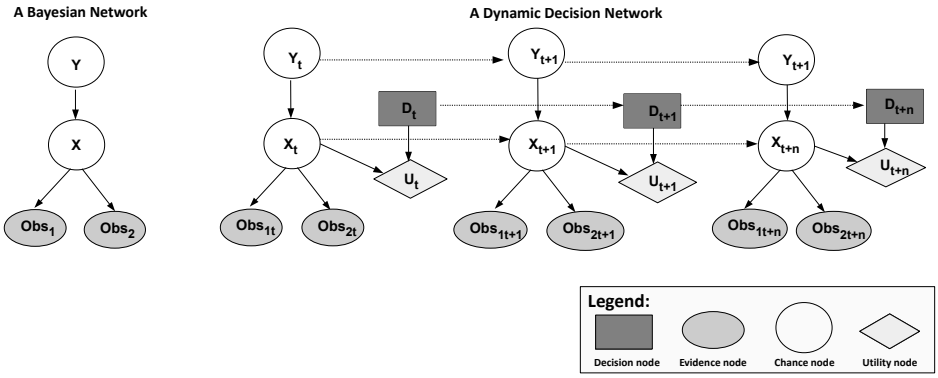


Figure 2. Structure of a DDN with dynamic chance nodes affecting utility nodes with decision nodes and evidences [34].

pervised learning (Bayesian models, decision trees, Markov models), unsupervised models (Neural networks), rules and fuzzy logic [25]. BNs are usually used to combine uncertain information from several sources to interpret high-level context information. Wolski et al. [40] presented the offloading decision as a Bayesian decision problem with a point of view of when to decide offloading under changing bandwidth using Bayesian theory arguing that in a Bayesian decision the inference of a new prediction is a well-defined function of the previously inferred prediction. Fenton and Neil [41] have used BNs for predictions of the satisfaction of non-functional aspects of a system. Esfahani et al. [42] employ fuzzy mathematical models to tackle the inherent uncertainty in their GuideArch framework while making decisions on software architectures. Dynamic configuration of service oriented systems was investigated by Filieri et al. [43] using Markov models. Many works use utility functions to qualify and quantify the desirability of different adaptation alternatives. These works are QoS-based, applied in different domains for resource allocation [44], typically in component-based mobile and pervasive systems such as Odyssey [45] and QuA [46].

Bayesian based models are well researched in multi-criteria decision making [41][47] as well and generally applied in clinical artificial intelligence [48]. Nonetheless, researchers [19][49][50] have investigated the feasibility and tractability of DDNs to solve the problem of decision making under uncertainty in self-adaptive systems. Bencomo et al. [19] used DDNs to deal with the runtime uncertainty in self-adaptive systems. In recent years, two optimization techniques have been developed to address dynamic decision scenarios: partially observable Markov decision processes (POMDP) and dynamic decision networks (DDNs). Both techniques are powerful enough and aim at solving complex, real-life problems that rely on the postulates of multi-attribute utility theory and probability theory. Costa and Buede [50] present a comparison between both the approaches and conclude that POMDP lacks an ability to achieve any tractable model [51], while DDN systems can present feasible solutions for complex cases. The value structure can not be replicated in an explicit way in POMDP for a multi-optimal decision environment. The current state of the system has to be known, in order to use Markovian decision models. It involves an extensive re-engineering effort to the system since its value structure is implicit in its every state and transition. Although the basic dynamics of POMDP are still Markovian, as the current state is not directly accessible, decisions require keeping track of (possibly) the entire history of the process, contrary to the Markovian property where there is no need to keep a track of all the previous states and observations to take a decision or perform an action [52][53]. In a DDN, all nodes that contain value objectives are explicitly connected to a utility/value node.

C. Learning the deployment trade-offs using DDNs

Conditional probability distributions (CPDs) derived by analysing historical attribute values helped solve stochastic problems in the past. The runtime setting for every component is hard to determine in advance due to the dynamic interaction of these components with the environment and the user. The adaptation module in our framework takes an advantage of probability theory and statistics to describe uncertain attributes. Probabilistic reasoning allows the system to reach rational decisions even when complete information is not available. We give a brief overview of the concepts of BNs and DDNs to understand the structure of the model used at runtime and how it is applied.

A BN is a Directed Acyclic Graph (DAG) that depends on Bayes' theorem [54] and CPDs. The graph is represented by a triplet (N, E, P), where N is the set of parameters, E is the set of arcs where each arrow declares the one parameter directly dependent on the one at the tail of the arrow, and P is the CPD for each parameter [34]. Figure 2 shows a BN with two chance nodes and two observation nodes. Chance nodes represent the influencing factors. BNs are able to reverse their inference logic due to the symmetry and usage of Bayes rule (given in Eq. (1)) and are able to update their beliefs on the fly as soon as a new evidence is observed [55]. Moreover, the Markov assumption (given in Eq. (2)) enables BNs and its dynamic counterparts, i.e., Dynamic Bayesian Networks (DBNs) to be fully operational even if an expert's opinion is fed to the model instead of an account of historic events.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

$$A^{t+1} \perp A^{(0:t-1)}|A^t \tag{2}$$

The computation of a posterior probability distribution over a model (or parameters) is called inference. It is one of the

TABLE II. Functional and non-functional (Qo*) requirements of the *Smart Lens* use case scenario.

Requirement	Description
Localization (QoS)	*Smart Lens* localizes the user within a building based on the camera. After recognizing a scene, it displays the position and orientation on a floor map and can give additional information about the user's surroundings.
Maximum Reliability (QoC)	The application should run reliably between the mobile and the cloud end without draining the mobile resources for other applications. A low reliability signifies that mobile is not the ideal execution environment, so the component should be deployed on the cloud.
Minimum Latency (QoS)	In MCC, the response time and performance of the application depends upon the location of the components and the amount of communication required to fulfil a task. Based on test data from CloneCloud [26], it should be reasonable to expect the application to return the results under 15 seconds.
Minimum Cost (QoC)	The switching cost between mobile and cloud impacts the performance of an application. It depends on the execution context, hence is considered a QoC requirement
Accuracy (QoS)	The results should be accurate. Because of the multitude of scenes that can be added to the dataset, it is also likely different locations will be represented by similar scenes, corridors and hallways.

basic operations offered by a BN. The precondition for inference is that the structure of the network/model is known and the prior probability distribution is already available. Learning can refer to the structure of the model, or the parameters, or both. Furthermore, learning may take place under either fully or partially observed variables. Learning offers a way to know the values of the parameters to properly explain the observed evidence. To represent parameters that change over time, it is possible to use a time-sliced network such that each time-slice corresponds to a time point in a form of a DBN. A BN/DBN does not necessarily require a historic data from one state to another for its inference, it can be suitable to perform the context reasoning for high-level context [56]. In order to realize proactive and situational decision making, two main concepts of Bayesian models are utilized, (1) Probabilistic QoS and QoC awareness using DBNs, and (2) dynamic decision making with DDNs to decide *when* to redeploy and *where*.

A DDN is a DBN that also includes a set of decision and utility nodes. The basic structure of a DDN is depicted in Figure 2. Decision node represents all the desired decision alternatives, connected to the utility node to compute its impact while making a decision. A chance node and other decision nodes can be the parents of a decision node. The utility nodes express the preferences among possible states of the world in terms of a subset of chance nodes and decision nodes. A probability-weighted expected utility is calculated for each decision alternative given the evidence. A chance node, decision node and even utility node can be expressed as its parents. The decision alternative is chosen according to the Maximum Expected Utility (MEU) principle. If a decision parameter *D* contains decision alternatives $\{d_1, d_2, ..d_n\}$ and *E* is the evidence parameter containing $\{e_1, e_2, ..e_n\}$ for a state parameter with states $\{s_1, s_2, ..s_n\}$, then the expected utility based on Bernoulli's equation [57] for taking a decision alternative is give as:

$$EU(d_j) = \max_D \sum_i P(s_i \mid E, d_j) U(s_i, d_j) \qquad (3)$$

We have employed DDNs to solve multi-objective, conflicting criteria problems while making Qo*-aware decisions over time for smart applications in MCC. Using a DDN is crucial in this research work for following reasons, given as:

- The type of relevant contexts evolve over time. Therefore, capturing the dependencies between the temporally variable relevance is difficult.
- If a static BN is employed, the interpretation of new evidence will lead to reinterpretation of previous evidence [58]. In order to overcome such a drawback, a DDN should be employed instead of a static BN.

Several intelligent applications are pretty lightweight, but others require a lot of computational effort (e.g., for prediction) or require analysis of large amounts of data (e.g., for pattern analysis). This work explores and utilizes the trade-offs involved in combining a data-intensive mobile application with context-awareness and cloud computing, and investigates the deployment of such applications in federated MCC environment. After identifying the deployment and performance trade-offs for outsourcing data and computation, our approach addresses the federation concern by continuously learning and adapting under multiple conflicting QoS and QoC objectives.

In the next section, we explain a use case scenario of a smart application, motivating the federated deployment of its components with our Qo*-aware framework.

III. Motivating scenario for the dynamic deployment decision

Advances in mobile technology accelerate the use of high-end data-intensive platforms on mobile devices using built-in sensors, such as an AR platform using the camera of the device. Mobile AR allows the devices to recognise objects. It requires extensive processing for image recognition and matching.

A. Use case scenario

We considered an indoor positioning use case application called *Smart Lens* [59] to investigate the relevant trade-offs motivating the runtime deployment requirements for context-aware applications in MCC. Context-aware intelligence is often found in AR applications to help the user explore certain places, be it cities, expositions, museums or malls.

In many cases, it makes the phone act as a camera but adds extra information next or onto objects that appear on screen. The core functionality of the application is to position the user based on the view of the camera. As soon as the application recognizes the scene, it displays additional information, such as a floor map, to give the user an idea of where he is as shown in Figure 3. Table II shows the functional and non-functional, i.e., Qo* requirements of the use case.

Smart Lens localizes the user within a building based on camera frames taken from a doorway. After recognizing a scene, it displays the position and orientation on a floor

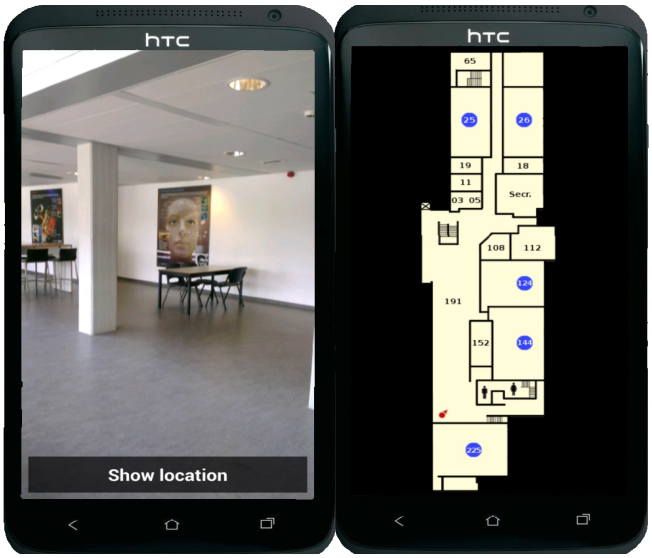


Figure 3. Scene recognition and image matching in the *Smart Lens*.

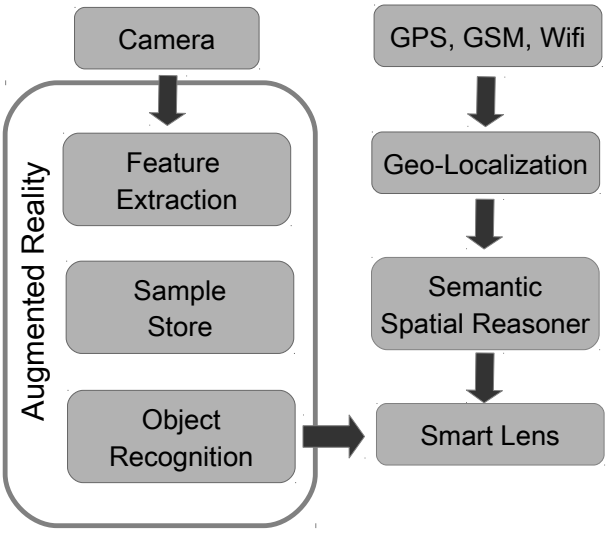


Figure 4. Overview of the components of *Smart Lens* use case.

map and can give additional information about the user's surroundings. The dataset can be large and continue to grow, adding more locations and poses, but can also be restricted to specific positions such as doorways to keep it practical. Modifiability can help the expansion of the dataset, which is having users capture and register their own scenes with the application. These users could be system operators, administrators of cooperating buildings, robots that explore the buildings or simply any user who wishes to register a location to navigate from or to. This application utilizes the location information of the user and the time of the day to further reduce the search space of objects to recognize and make the comparison smarter. We have analysed the performance and resource utilization trade-offs for our use case, motivating the need of the smart offloading decision. The next subsection discusses few of them.

B. *Trade-off analysis*

The AR components *Feature Extraction* and *Object Recognition* components require a relatively large amount of processing time, which not only drain the battery further but causes the application and the phone as a whole to slow down as well. Figure 4, provides an overview of the composition of the components for the *Smart Lens*. The resource and performance trade-offs are investigated for AR components by analysing a correlation between the latency of recognition, the structure of the dataset, i.e., its size and the quality of the images. Communication and computation trade-offs are also analysed for our use case.

The mobile device is held up facing an exact copy of a sample in the set. The use case application is started and shortly after, the time until recognition of the scene is printed on the screen. It is repeated for datasets of size 3, 50, and 100, consisting of high quality samples, low quality samples or a mixture of both. The size determines against what number of samples each scene needs to be compared, and the quality and detail in the samples themselves affect the amount of work needed for such comparisons.

Two different experiment scenarios are conducted: 1) when dataset is stored on the mobile device and the AR components perform the computations locally on the device, 2) when the cloud deployment for AR components is used keeping the entire dataset on the remote location as well.

1) Performance vs dataset size on the mobile: The results in Figure 5 demonstrate the effects of the dataset structure (on x-axis) on the recognition latency (on y-axis). Choosing the right dataset has an important influence on the performance. Larger local datasets slow down any data intensive scenario, in our case image recognition and AR based application requiring more resources impacting the performance negatively, and the use of high quality feature rich samples slows down detection when faced with an ideal scene.

However, it should be noted that smaller sets are less likely to contain the correct sample and require more computations if the applications is used for multiple detections, and lower quality samples have considerably more difficulties detecting scenes from a perspective that is not the same as the one when capturing the sample. Similarly, low quality samples also reduce the file size of the dataset. A smaller set not only reduces latency, calculation time and therefore energy consumption, it also reduces the file size, lowering the need for memory and possibly data communication. Smaller datasets can be obtained by filtering all samples in the system to those that are plausible given the current context of the device and the users. Additionally, devices with limited resources can choose to use datasets of lower quality, if offered, to save energy and calculations while sacrificing performance, hence instating the decision making on trade-offs.

2) Performance vs dataset size on the cloud: Figure 6 shows the latency when the dataset is placed on the cloud and the computation is done on the cloud. Lower latency with Wi-Fi shows that the performance of the application now improves with better Internet connectivity, as could be expected. It is not possible to offload fewer computations without fetching a part of the dataset, requiring additional communication with remote servers and anticipation and filtering of the dataset. AR applications using context-awareness to filter datasets, results in less memory usage and lower detection latency, whereas with cloud computing detection latency is approximately constant with respect to the dataset size, and no local memory is occupied

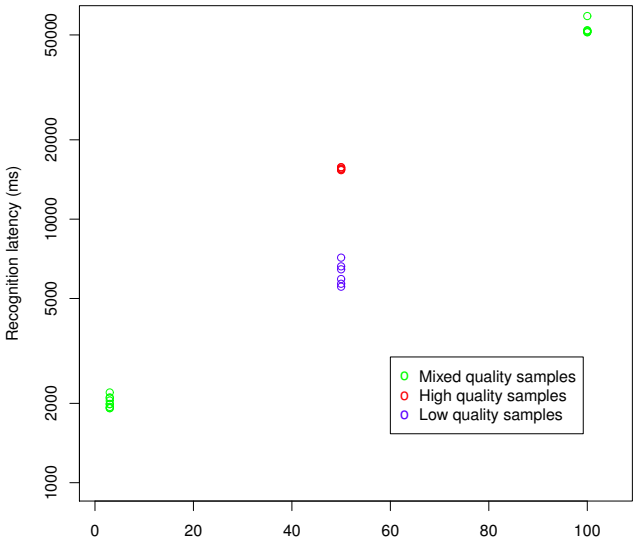


Figure 5. The latency in milliseconds (scaled logarithmically), when the recognition is performed on the mobile device with local dataset.

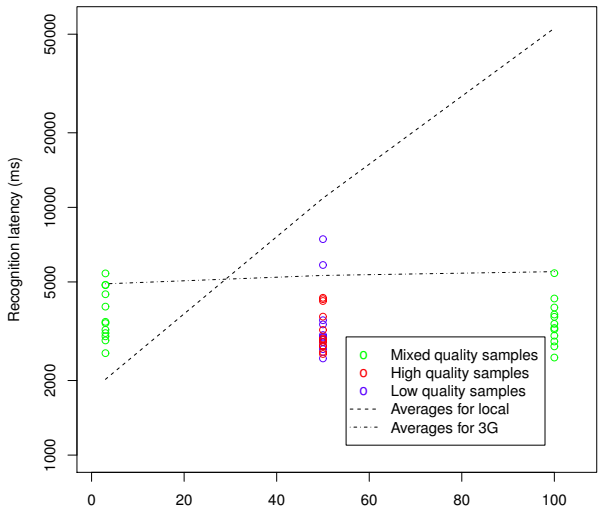


Figure 6. The latency in milliseconds with AR components and the dataset are offloaded to the cloud and accessed via WiFi. The trade-off curves of latency average for datasets stored on mobile and accessed via 3G are shown by the dashed lines.

by the dataset. But the latency due to connectivity type and the amount of data to be communicated is a major trade-off. The trade-off curves for our use are also shown in Figure 6. Placing the dataset on the cloud servers with the computation intensive components, leads to a need for a smart computation offloading method on the mobile device that learns the resource utilization and performance trade-offs in order to dynamically deploy the components between mobile and cloud infrastructure.

We will discuss the research requirements for automated decision support for the dynamic deployment considering the above mentioned trade-offs.

IV. Requirements for Qo*-aware smart deployment decision

A modular design philosophy for data and control processing on the mobile and cloud ends is needed to achieve a flexible distributed deployment. It simplifies redeployments and reconfigurations significantly. In our previous work [60], we demonstrated that a modular application design philosophy helps to support optimal mobile cloud application deployments. Moreover, we identified that many resource and performance trade-offs exist [13] in such a federated deployments for other use cases as well. The large number of parameters associated with the deployment configurations for these applications make it nearly impossible for developers to fine tune them manually. Therefore, automated deployment and optimization are necessary [61].

Many opportunities for optimization exist as there are several distributed deployments of the application components and different configurations per component possible. The challenge is to find and analyse different optimization trade-offs in a federated environment of MCC, each characterized by varying sensing, communication, computation and storage capabilities. Furthermore, addressing the influence of runtime uncertainty in the context and its quality is an utmost essential task.

The investigated research requirements for a Qo*-aware deployment framework are given below:

1) **Offer reflective decision support:** The decision maker should be able to decide for which execution scenario the cloud is better and for which ones a cloud based deployment does not bring any added value, addressing *when* to offload in an automated way. Generally, the adaptation decision includes the ways to split responsibilities between devices, applications and the cloud servers. Since last decade, MCC research focus is to develop algorithms and techniques for deciding *whether* offloading would be beneficial or not and *what* to offload [62]. With continuous improvements in connectivity and mobile technology, the focus is shifting to smart offloading where the decision support is required with an aim of *when* to use MCA. Context-aware applications generate data at a large rate and sometimes in an uncontrollable fashion. The essence of the problem is finding the middle ground, determining what the mobile devices should handle and what the cloud should handle. Sending the raw data to the remote cloud servers is not always feasible [13]. Moreover, smart offloading demands a reflective decision support where the decision maker can look-ahead for the impact of the current decision and predict its impact on future situations.
2) **Process and provision the context:** As context is an essential constituent of smart applications. Runtime Context provisioning is inevitable in such a federated decision making. It always incurs a cost to profile the resources or other context parameters in mobile runtime environment. Furthermore, the continuous varying context and processing of the heterogeneous sensory data introduce challenges to take the most rational decision.
3) **Process the Quality-of-Context (QoC):** QoC raises more questions while dealing with automated adaptation. Context provisioning is a multi-level process [36] where low-level events are enriched through filtering and aggregation. For example, GPS coordinates are read from the sensor and translated into a high-level description of the location of the user. Finally, the desired high-level context is inferred

TABLE III. Objectives of the Qo*-aware offloading in MCC.

Objective	Description
Qo*-awareness	A traditional QoS & QoC requirements gathering to identify and model the required quality attributes at design time. It is domain specific and involves the type of context being utilized. The system should be able to capture the real-time context of the user and his environment.
Self-Adaptive	The system must be self-adaptive at runtime to optimize the resource consumption of the application by outsourcing few components to the cloud.
Optimize Offloading	The system should detect the runtime context of the mobile device, i.e., CPU usage, memory consumption and battery usage. Runtime support to detect a change in QoC in a particular context type and optimize the offloading for Qo* requirements while performing opportunistic offloading. System should measure its impact on other context types before making any decision. Runtime support to detect a change in QoS before making any decision.
Resolve conflicts	The system should select the deployment strategy with respect to QoS & QoC requirements from the application's perspective in a total qo*-aware fashion, such as *Maximum Reliability*, *Minimum Latency* and *Minimum Cost* should be met in our use case.

through context reasoning. For instance, presence of the user can be inferred by his/her location. The adaptation decision is based upon this high-level context. End-to-end QoC control is a safeguard to monitor the quality of the context data throughout this multi-level process. The framework should consolidate the requirements of QoC in the context provisioning under varying context.

4) **Maintain the Quality-of-Service (QoS):** The re-configuration adaptation needs to be performed in an efficacious way without hindering the QoS of the applications. The possible re-configuration variants of any application can be determined at design-time in a static way but to achieve the multiple-objectives in MCA, it highly depends on the varying situation at runtime introducing the need of a decision support at runtime that does not hurt the QoS requirements of the smart applications in all possible deployment scenarios. This imposes a big challenge, especially for resource-limited mobile devices, when conflicting optimization objectives are involved. The decision maker should not only process the context but also predict the effect of the decision for future QoS requirements.
5) **Display retrospective behaviour for trade-offs in federated deployments:** Ascertaining the quality of the decisions is an important aspect in order to meet the QoS requirements that are directly affected by the context and its quality for smart applications.

Context data is generated depending on the objectives of any smart application and its size varies accordingly. If the decision maker decides to achieve a certain requirement, obviously it has to compromise on another aspect. It is non-trivial for the decision maker to learn and memorize the decisions taken. Attaining a retrospective behaviour for MCC federated deployment decision is challenging due to the constrained mobile environment and the overhead has to be taken into account.

Table III shows the overall objectives of our framework according to the above mentioned requirements. An abstract overview of the QoC-aware adaptation is shown in Figure 7. It depicts the flow of end-to-end QoC and its processing at each level of the context-processing. The dotted area shows the significance of QoC-awareness in decision support using probabilistic models. We have used probabilistic models to mitigate the runtime uncertainty in the available context. In the next section, we will explain our Qo*-aware dynamic deployment framework and its learning mechanism using probabilistic models to achieve well-informed and reflective decision making with the above described features.

V. Context-driven dynamic deployment approach

Our framework consists of a loosely-coupled context-processing system along with an adaptation module. As shown in Figure 8, mobile hosts a *Dynamic Adaptation Module*, i.e.,

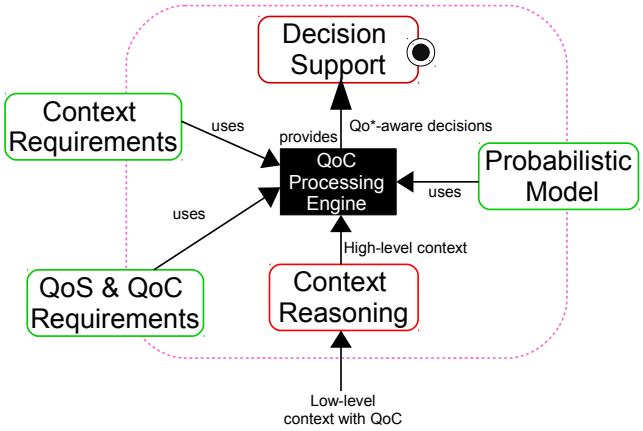


Figure 7. An abstract view of QoC-aware decision support in our framework.

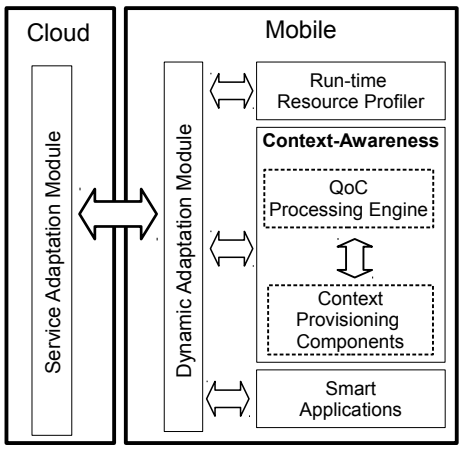


Figure 8. A blueprint of our federated framework and its dynamic deployment modules.

a component running on the mobile device to adapt the deployment configuration of the applications. However, the cloud environment hosts a *Service Adaptation Module*, which aims to optimize the runtime deployment of the required components and acts as an entry point for the adaptation module on mobile client. This module receives raw or pre-processed context data (including the type of the content and the identity of the source) and forwards it to a publish/subscribe subsystem so that interested parties (i.e., the subscribers) receive context updates. *Dynamic Adaptation Module* uses the model@Runtime approach and hosts the DDN model for adaptation. The next subsection details the working procedure of this module.

A. Deployment and reconfiguration decision making

Deployment Adaptation Module takes the decision of how to split responsibilities between mobile devices, applications and the framework itself. It is able to decide on the opportunistic use of the cloud. Our decision making approach for redeployment and reconfiguration is explained in the steps mentioned below:

Information discovery and selection – The framework discovers and explores the application's runtime environment in order to get the context information to work with. Figure 9 shows a taxonomy of the runtime environment of a mobile device. Its resources can affect the ability to meet the QoS requirements and eventually, the decision of dynamic deployment. Our framework discovers the sensors and context types required for the smart application. Built-in sensors in mobile devices are important to fetch the context data, but the size of the acquired context data varies, depending on the application's objectives. The framework filters the acquired context according to specific needs of the application. This selection process can be fairly complex as it may require complex filtering techniques to decide which sensor or device is offering relevant information. QoS and QoC requirements are gathered at this step to bootstrap the decision making. *Runtime Resource Profiler* component deployed on the mobile side, gathers these requirements. The utility structure of the system is maintained here corresponding to the QoS requirements of *Minimum Latency* (L_{min}) and QoC requirement of *Maximum Reliability* (R_{max}) for the mobile device.

Analysis and decision making – The inferred context information is used to bootstrap the deployment decision or to change the configuration or behaviour of the application. System does a probabilistic analysis among the conflicting objectives in order to achieve the QoS and QoC requirements while making a decision. The computation or storage resources on the device and the communication cost affect the decision. For instance, the user points his mobile device on a scene. The image/video frame from the mobile device is captured. The feature points for the image in the frame are extracted. With these feature points, scenes are then identified by matching the extracted feature points to those of known scenes in a database. There is little memory because of his running video player. In this situation, a thin client configuration is chosen for *Smart Lens*, delegating *Spatial Reasoning* and *Object Recognition* components to cloud infrastructure. Furthermore, when the user shuts down the video player, a context change is raised: the free memory on the hand-held increases.

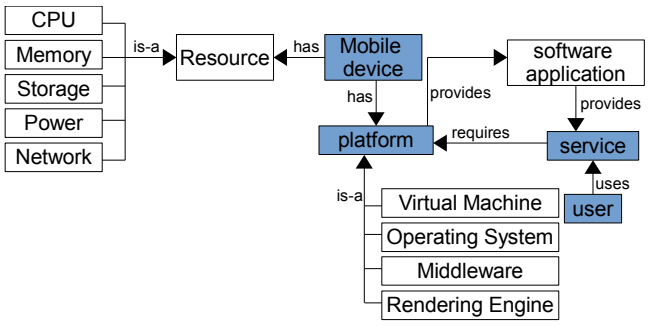


Figure 9. Taxonomy of the runtime environment of a mobile.

Enactment of the decisions – With a thin client configuration as in the previous step, the framework has to observe the real-time impact of the configuration to maintain QoS requirement of L_{min}. In order to decrease the application response time *Smart Lens* is reconfigured with a thick client and caching of data to save power and to become less vulnerable to network instability. The decision making is a continuous process, where the framework optimizes the application behaviour to reach certain objectives on the basis of the required attributes. When the availability of required resources varies significantly, the framework has to decide whether to trigger an adaptation in the form of reconfiguration of the components. It learns from its previous decisions and the available context in order to ascertain the quality of the decision and learn from its own mistakes to achieve better results in the future. The adaptation modules on both the ends communicates with the *QoC Processing Engine* for the deployment decision support. The context provisioning services use the *QoC Processing Engine* in order to provide Qo*-awareness.

B. Model structure

We present the details of the model structure and the involved multi-criteria parameters in this section. A DDN is modelled for the enactment of the decisions that change over time influenced by dynamic states and preferences. The first step is to identify the involved uncertain parameters and the causal relationships between those parameters [47]. Extensive interaction with the domain experts is vital to structure a quality model in order to fulfil the requirements. Table IV shows the identified parameters for our problem domain and their nature based on the requirements of the use case (see Section III). The value of the static variables is independent of their counterpart in multiple time slices. Dynamic parameters are affected in multiple time slices by their historic values. The effectiveness of reconfiguration decisions over time are investigated for multiple consecutive time slices. Each time slice contains an action taken by the system.

TABLE IV. Parameters types and values for QoC*-aware dynamic deployment.

Parameters	Values	Nature	Availability
CPU Usage	high, low	static	observed
Memory Usage	high, low	static	observed
Data Storage	remote, local	static	provided
Available Bandwidth	low, high	static	observed
Available Resources	yes, no	static	inferred
Required Connectivity	yes, no	static	inferred
Maximum Reliability	low, medium, high	dynamic	inferred
Minimum Switching cost	low, high	dynamic	inferred
Minimum Latency	minimum, average, maximum	dynamic	inferred

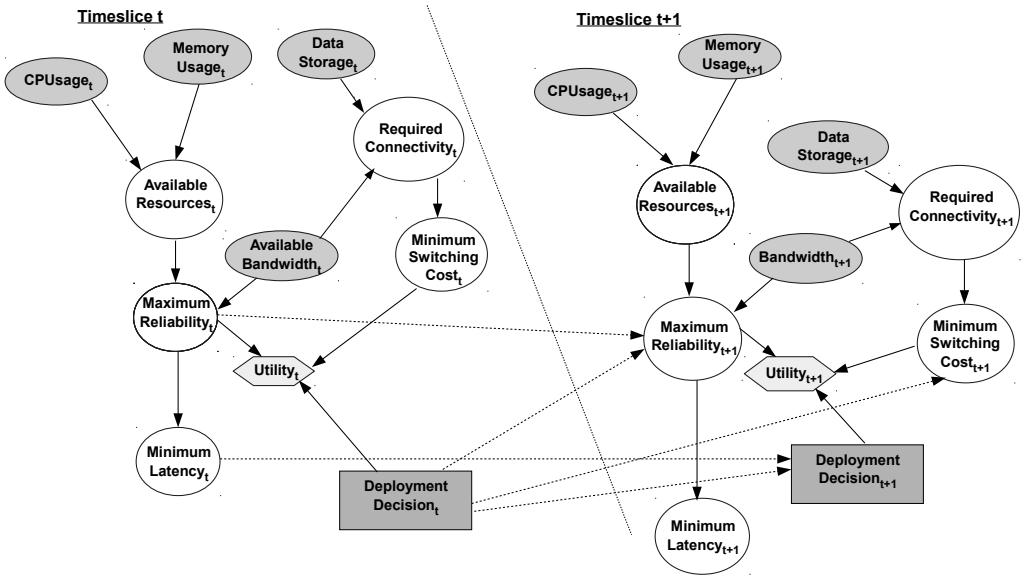


Figure 10. DDN model for dynamic deployment domain expanded in two time slices.

We have designed a DBN model to tackle the requirements 2, 3, and 4 identified in Section IV and a refined in the form of a two time sliced DDN model to address the rest of the requirements, i.e., 1 and 5 from Section IV. Figure 10 shows our DDN model.

The *Deployment Adaptation Module* uses this network to decide about the dynamic redeployments for the smart application components. We modelled the decision as a finite-horizon, sequential decision process [34]. At each time slice, the *Deployment Adaptation Module* decides on the fly about a component, whether to put it on the mobile or it should be running on the cloud. The time slice corresponds to a change in the context values of the execution environment of the mobile device depending upon the profiling interval. The use and feasibility of the DDN models are evaluated in [19][49] for other self-adaptive domains.

Our model expresses QoS and QoC requirements (*REQ*) by chance nodes and these requirements are causally linked by the involved context expressed as the observation nodes as shown in Figure 10. These chance nodes make a BN with the CPDs corresponding to the effects of *Deployment Decision* D_j alternatives $\{$*mobile, cloud*$\}$ over conflicting *REQs* $\{$*Maximum_Reliability, Minimum_Cost*$\}$ expressed as *P(REQ$_i$ | d$_j$)*. *Available Resources* is a context parameter that stochastically varies according to the runtime environment parameters, i.e., *CPU usage* and *Memory* bringing uncertainty. *Bandwidth* is a random parameter observed dynamically. *Maximum Reliability* is the QoC parameter as it is inferred from the *Available Resources* on the mobile device and *Available Bandwidth* information, playing a vital role in decision making. Its value *low* shows that mobile device is not a reliable execution environment for the components, therefore, *Maximum Reliability* effects the utility of the decision. *Minimum Switching Cost* is another QoC parameter casually linked with *Data Storage* to capture the communication trade-offs while taking a *Deployment Decision*. The QoS parameter *Minimum Latency* is effected from the QoC parameter *Maximum Reliability* and plays a vital role in decision prediction, hence it is causally linked with the *Deployment Decision* in future time slice.

C. Utility function

A utility function computes the subjective choices of a decision maker for available decision alternatives and its outcomes. Elicitation of utility values requires the knowledge of the involved subjective probabilities [63]. In Bayesian inference, a decision is determined with both the utility function and the posterior probabilities. The utility values can be obtained by the domain experts or from the decision making preferences. We assigned the utility function for each of the preference criteria same as defined in [47]. We applied a linear transformation to normalize the utility values in order to reduce the computations for our DDN. The normalization formula is given as [47]:

$$U_i = 1 - \frac{V_{max} - V_i}{V_{max} - V_{min}} \tag{4}$$

The normalized utility values are given in Figure 11 for the most desirable decision alternative in the range from 1 to 100. For each *REQ$_i$*, the utility nodes express the utility function that takes the CPDs of the *REQs* and their priorities into account. *U(REQ$_i$ | D$_j$)* represents the numerical weight of each requirement and *P(REQ$_i$ | E, D$_j$)* represents the conditional probability for each *REQ* under the current observed context (evidence) *E* $\{$*CPU Usage, Memory Usage, Available Bandwidth, Data Storage*$\}$. The expected utility [34] for each decision is computed by Eq. (5) and the decision is chosen by the MEU principle.

$$EU(D_j \mid E) = P(REQ_i \mid E, D_j)U(REQ_i, D_j) \tag{5}$$

VI. Experimental Evaluation

In this section, we discuss the experimental setup and the results obtained towards an opportunistic offloading decision support using DDNs.

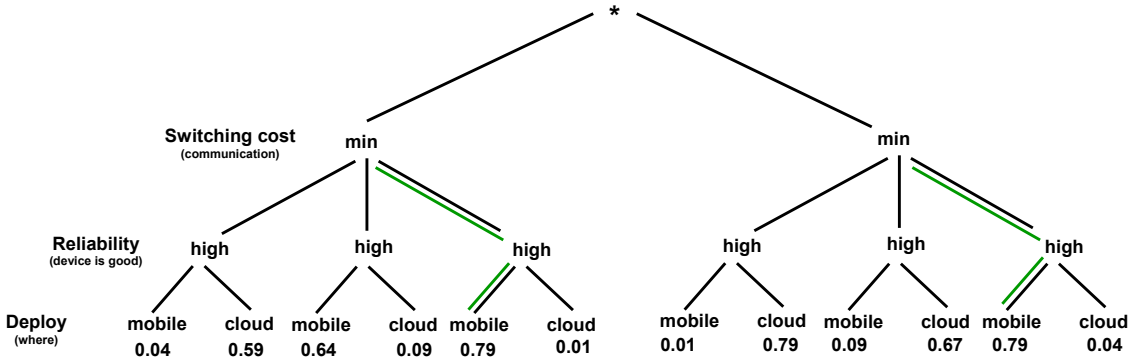


Figure 11. The numeric weights assigned to the conflicting objectives, the most favourable path is highlighted with a green line.

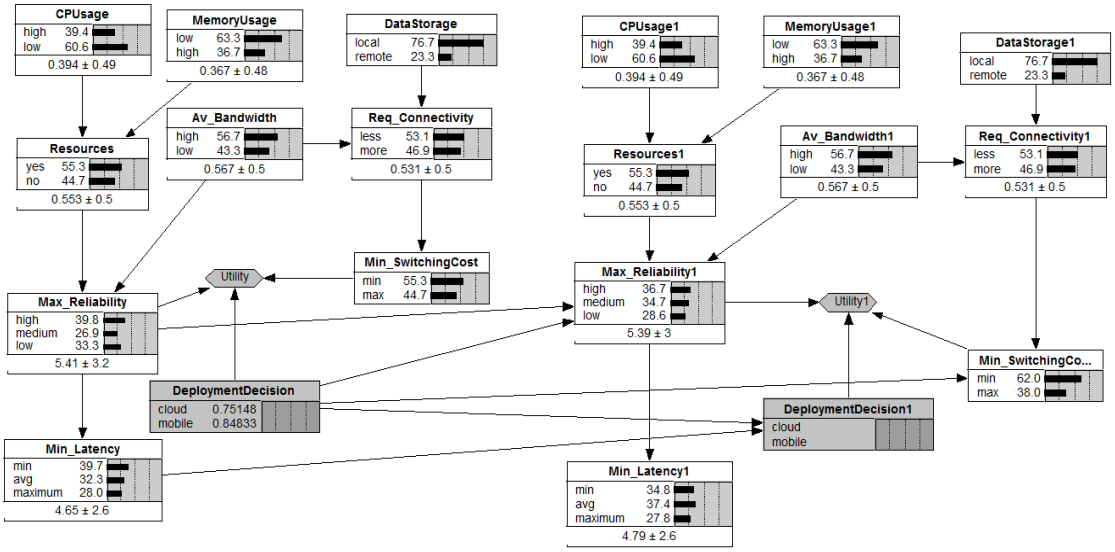


Figure 12. Prior probabilities of our DDN in two time slices.

A. *Enabling technologies & implementation*

In order to implement our framework, discussed in Section V-A, we are using an HTC One X with a 1.5 GHz Quad Core ARM Cortex processor (at about 2.5 MIPS per MHz per core) to run the Android-based *Smart Lens*, augmented reality application, which embeds the Vuforia library to recognize any scene.

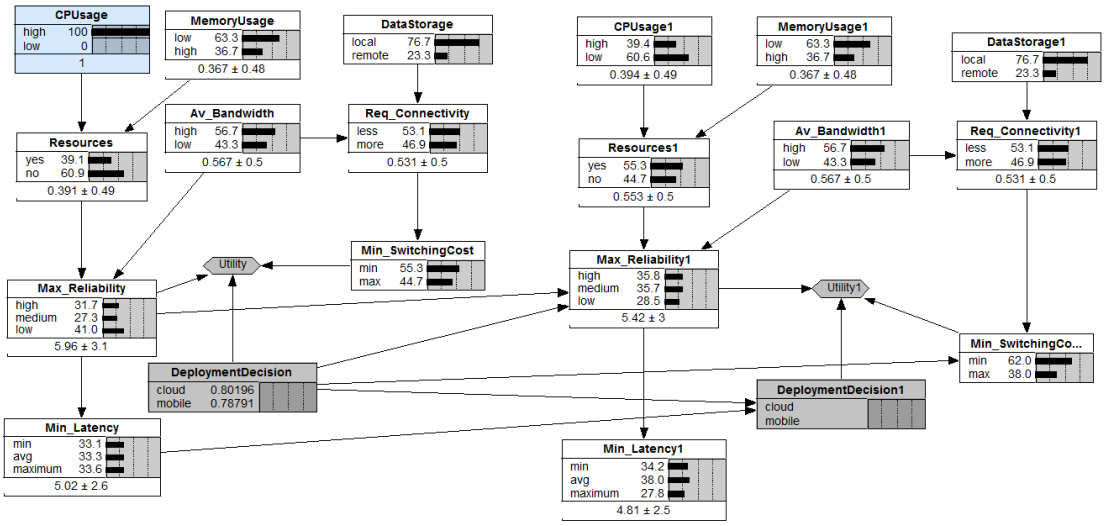
Our infrastructure runs VMware's open source Platform-as-a-Service (PaaS) offering known as Cloud Foundry on a server with 8 GB of memory and an Intel i5-2400 3.1 GHz running a 64-bit edition of Ubuntu Linux 12.04. A Java-based implementation has been used for *Runtime Resource Profiler* that captures the runtime context of mobile device. We have modelled a DDN-based probabilistic model for the components of our *Smart Lens* use case using the Netica development environment (http://www.norsys.com). We give a detailed account of the experiments conducted in Netica to analyse the deployment adaptation decisions in our model.

B. *Belief propagation and computation of the expected utilities*

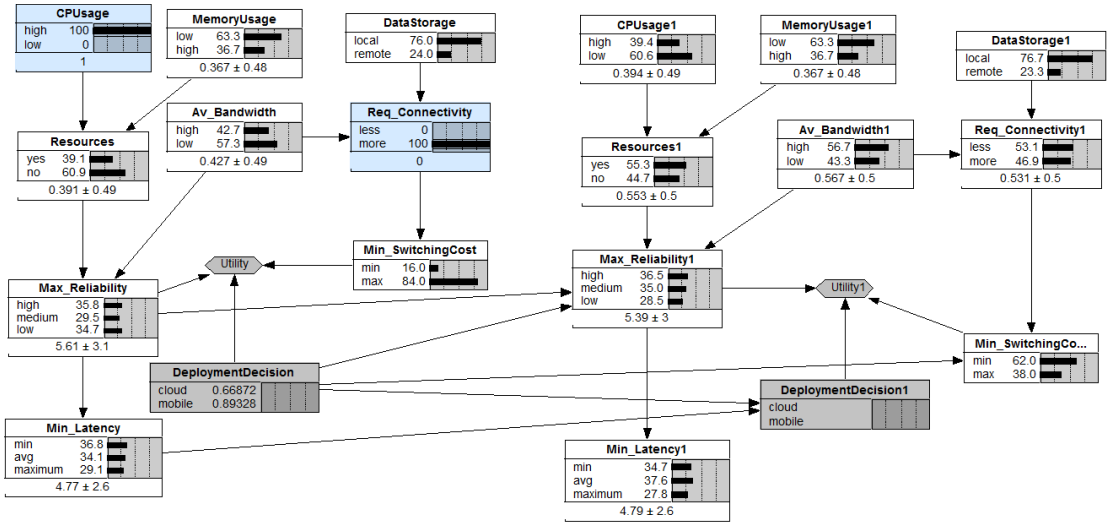
We investigated the belief propagation in our DDN model to analyse the decisions taken by it under changing context of the mobile device. The model can be bootstrapped with prior probabilities. These probabilities are learnt from experience dataset or can be set by domain experts in the same way as the policies can be set for rule based systems. Figure 12 depicts the initial computation of our model bootstrapped with the prior probabilities set according to the domain requirements. Under favourable conditions for execution environment, it can be seen that the decision to run the component on *mobile* has higher expected utility. As context is generated from mobile device, the prior probability for *Data Storage* is set to *local*. These prior probabilities are overwritten by the evidence in the form of observations from mobile resource profiling for belief propagation in the network. The experiments are conducted for the redeployment of *Object Recognition* component, whenever the execution environment changes on the mobile device.

In our first scenario, we observed the CPU usage as high with all the other parameters uncertain, the expected utility for the decision changes on the basis of the inferred *Maximum Reliability* as the *Switching Cost* remains *min* (see Figure 13a). When the *Switching Cost* changes to *max*, the expected utility is recalculated and it is triggered to choose *mobile* as a deployment option (see Figure 13b). This experiment shows that our model can effectively cope with the conflicting trade-offs and choose a favourable decision. In the second time slice, state of the Qo*-parameters are predicted and analysed. Once the decision is chosen it updates the decision for the future and recompute the belief propagation as shown in Figure 13c.

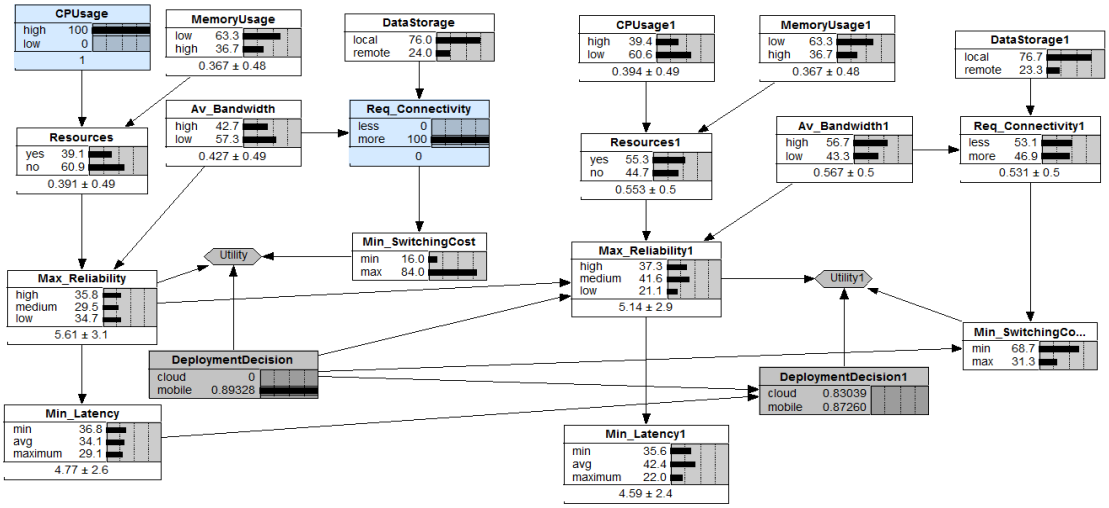
To evaluate the proactive adaptivity of our DDN model, we



(a) Expected utility with *CPU Usage* observed as *high*.



(b) Expected utility with *CPU Usage* observed as *high* and *Switching Cost* observed as *max*.



(c) Predicted state of the Qo*-parameters and beliefs in future.

Figure 13. Belief propagation in our DDN model for dynamic deployments on the fly.

TABLE V. Evaluation of our DDN model for different dynamic deployment

Expected utilities (EU)		Context parameters		Qo*-awareness parameters			Decision
Mobile	Cloud	Resource	Available bandwidth	Reliability on mobile	Switching cost	Performance	max(EU)
0.664	0.958	yes but with high CPU	high	high	min	avg	cloud
0.636	0.928	no	high	very high	min	avg	cloud
0.427	0.396	prior belief	prior belief	prior belief	prior belief	max	mobile
1.089	0.505	prior belief	low	low	max	max	mobile
1.006	0.563	no	high	high	max	avg	mobile

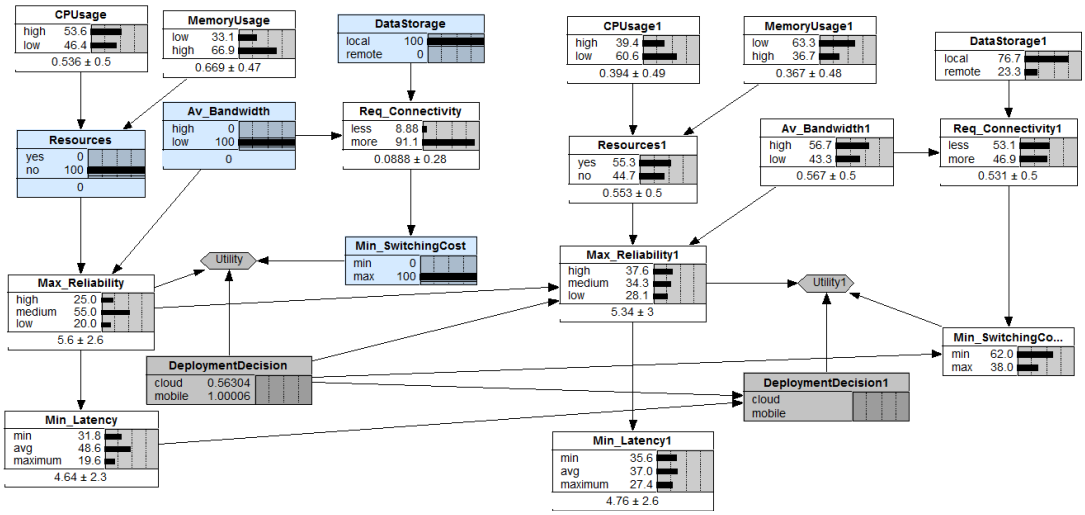


Figure 14. Expected utilities when there are no resources available on mobile device.

have conducted several experiments with the changing context environment on the mobile device. Table V shows the scenarios and the Maximum expected utility for these scenarios. Figure 14 depicts a snapshot of all the parameters and their beliefs. The *Maximum Latency* value is always analysed in the second time slice to achieve a proactive behaviour. In our first experiment, an automated trade-off analysis is done before choosing the cloud as the *Available Bandwidth* is high but *Resources* are on a low side. The decision model triggers an adaptation decision on the basis of the context parameters for mobile execution environment and do a trade-off analysis for Qo*-awareness. The most favourable decision alternative is chosen, as evident from the results in every case. Figure 15 shows the results for these experiments and chosen decision alternative for each runtime setting. The third scenario in the graph shows the intelligence of our model when there is no information available and it makes a decision in total uncertainty based on the prior beliefs.

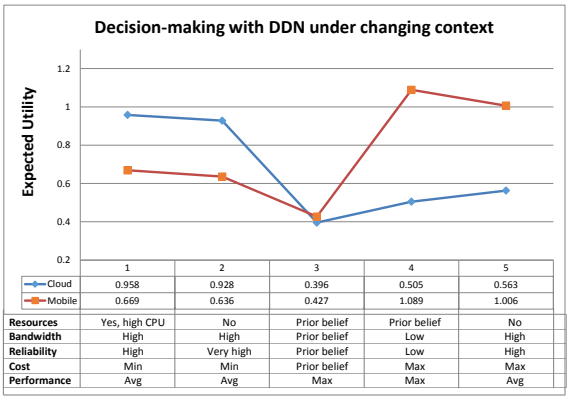


Figure 15. Dynamic decision making for the changing context on the mobile device.

We have conducted several experiments with and without stressing the CPU, in order to evaluate the performance overhead of processing a DDN-based model for deployment decision making. The performance overhead of running a 2-sliced DDN on a Samsung S4 Android device is 5.8 milliseconds without stressing the CPU. But if the CPU is busy and stressed, the processing time goes up to 6 milliseconds. There is an overhead involved due to the evidence collection for runtime context. The processing with stressed CPU on the same device goes up to 16 milliseconds with an overhead of 10 milliseconds for the evidence collection.

VII. Conclusion and Future Work

MCA addresses the challenges to the resource limitation for mobile devices preserving the QoS requirements. Deployment for the components of smart applications in MCA is a non-trivial task in the presence of many resource-performance trade-offs and compromises. These compromises can affect the QoS or QoC for context-aware applications. The optimal strategy to deploy and configure intelligent applications with dynamic and heterogeneous resource availability cannot ignore the interplay between QoS and QoC. A modular design philosophy for developing intelligent applications helps to dynamically configure, compose and deploy these components. The overall aim of our work is to intelligently automate the distributed deployment and configuration of the components across the mobile and cloud infrastructure, and to realize an opportunistic use of the cloud.

In this paper, we have presented a novel approach for dynamic deployment decision making in federated environment of MCC by leveraging DDN to automate decisions in a continuously evolving runtime environment context. DDNs build upon DBNs. However, the latter is only able to learn

conditional probabilities based on a dataset, whereas DDNs can quantify the impact of the evidence and the effect of the decisions. Furthermore, by exploiting the utility of deployment decisions, our framework can learn how to automatically improve its decisions for future. Our first contribution is an analysis for the involved trade-offs for smart applications. To cope with the trade-offs for quality-aware deployment decisions, we present a Qo*-aware decision making framework based on DDNs in MCC domain. A feasibility analysis of incorporating DDNs for decision making was performed, and our experiments have clearly demonstrated the ability of adapting its decision in the presence of evolving situations and an uncertain context of the environment. By incorporating QoS and QoC in our DDNs, we are able to assess the quality of our context-driven decisions, ascertain their quality and update future decisions and corresponding actions according to the outcome and impact. Our experiments have shown that an intelligent application can achieve optimal deployment for its components under a reasonable overhead, whenever the context is updated. However, the overall success of the model highly depends on the subjective probabilities and the utility function and its values. The sensitivity analysis [19][39] of these models validates their dependency on the prior beliefs and the utility values.

Applications of probabilistic theory and other artificial intelligence techniques can help to achieve the real meaning of smartness in several domains particularly in MCC. The limited tool support for their application is indeed a hurdle to widely utilize these techniques. We are actively working on our framework to realize the practical use of DDNs for dynamic deployment purposes in MCC using different available platforms for Bayesian inference and DDN support. In our work, we used two time slices network but we are interested to conduct a performance analysis of DDNs on a mobile device where it can be investigated that how many time slices are important in order to practically utilize these models in mobile environment without creating an overhead on device resources. Further work is required towards more systematic techniques for the runtime synchronization of multiple DDN models and to empirically study the scalability of these models. The value of the probabilities that change over time and their impact on alternative decisions can also be of interest. Finally, developing tools to specify the QoC requirements would be certainly very helpful as current tools support is fairly limited.

ACKNOWLEDGMENT

This research is partially funded by the Research Fund KU Leuven.

REFERENCES

[1] N. Z. Naqvi, D. Preuveneers, and Y. Berbers, "Dynamic deployment and reconfiguration of intelligent mobile cloud applications using context-driven probabilistic models," in *INTELLI 2014, The Third International Conference on Intelligent Systems and Applications*, pp. 48–53, 2014.

[2] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[3] E. H. Aarts and S. Marzano, *The new everyday: Views on ambient intelligence*. 010 Publishers, 2003.

[4] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*, pp. 304–307, Springer, 1999.

[5] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[6] T. Buchholz, A. Küpper, and M. Schiffers, "Quality of context: What it is and why we need it," in *Proceedings of the workshop of the HP OpenView University Association*, vol. 2003, Geneva, Switzerland, 2003.

[7] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[8] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution.," in *HotOS*, vol. 9, pp. 8–11, 2009.

[9] Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[10] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pp. 4–4, 2010.

[11] OASIS, *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*, 2013 (accessed May 23, 2015). Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[12] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 337–368, 2014.

[13] N. Z. Naqvi, D. Preuveneers, Y. Berbers, *et al.*, "Walking in the clouds: deployment and performance trade-offs of smart mobile applications for intelligent environments," in *Intelligent Environments (IE), 2013 9th International Conference on*, pp. 212–219, IEEE, 2013.

[14] D. Chalmers and M. Sloman, "A survey of quality of service in mobile computing environments," *Communications Surveys & Tutorials, IEEE*, vol. 2, no. 2, pp. 2–10, 1999. [retrieved: October, 2013].

[15] N. Chen and A. Chen, "Integrating context-aware computing in decision support system," in *Proceedings of the International MultiConference of Engineers and computer Scientists*, vol. 1, 2010.

[16] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[17] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press, 1998.

[18] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, *et al.*, "Software engineering meets control theory," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015.

[19] N. Bencomo, A. Belaggoun, and V. Issarny, "Dynamic decision networks for decision-making in self-adaptive systems: A case study," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, (Piscataway, NJ, USA), pp. 113–122, IEEE Press, 2013.

[20] C.-Y. Ting and S. Phon-Amnuaisuk, "Factors influencing the performance of dynamic decision network for inqpro," *Computers & Education*, vol. 52, no. 4, pp. 762–780, 2009.

[21] L. Portinale and D. Codetta-Raiteri, "Using dynamic decision networks and extended fault trees for autonomous fdir," in *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pp. 480–484, IEEE, 2011.

[22] A. Y. Tawfik and S. Khan, "Temporal relevance in dynamic decision networks with sparse evidence," *Applied Intelligence*, vol. 23, no. 2, pp. 87–96, 2005.

[23] T. Charitos and L. C. Van Der Gaag, "Sensitivity analysis for threshold decision making with dynamic networks," *arXiv preprint arXiv:1206.6818*, 2012.

[24] A. Khan, M. Othman, S. Madani, and S. Khan, "A survey of mobile cloud computing application models," *Communications Surveys Tutorials, IEEE*, vol. 16, pp. 393–413, First 2014.

[25] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 414–454, 2014.

[26] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314, ACM, 2011.

[27] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.

[28] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," in *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, pp. 233–248, ACM, 2012.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Unleashing the power of mobile cloud computing using thinkair," *arXiv preprint arXiv:1105.3232*, 2011.

[30] D. Narayanan, J. Flinn, and M. Satyanarayanan, "Using history to improve mobile application adaptation," in *Mobile Computing Systems and Applications, 2000 Third IEEE Workshop on*, pp. 31–40, IEEE, 2000.

[31] G. Huerta-Canepa and D. Lee, "An adaptable application offloading scheme based on application behavior," in *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pp. 387–392, IEEE, 2008.

[32] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, pp. 59–79, Springer, 2012.

[33] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010.

[34] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*, vol. 74. Prentice hall Englewood Cliffs, 1995.

[35] S. Chen, J. J. Lukkien, and P. Verhoeven, "Context-aware resource management for secure end-to-end qos provision in service oriented applications," *Journal of Ambient Intelligence and Smart Environments*, vol. 3, no. 4, pp. 333–347, 2011.

[36] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," *ACM Computing Surveys (CSUR)*, vol. 44, no. 4, p. 24, 2012.

[37] K. Sheikh, M. Wegdam, and M. v. Sinderen, "Quality-of-context and its use for protecting privacy in context aware systems," *Journal of Software*, vol. 3, no. 3, pp. 83–93, 2008.

[38] Y. Kim and K. Lee, "A quality measurement method of context information in ubiquitous environments," in *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, vol. 2, pp. 576–581, IEEE, 2006.

[39] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[40] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, IEEE, 2008.

[41] N. Fenton and M. Neil, "Making decisions: using bayesian nets and mcda," *Knowledge-Based Systems*, vol. 14, no. 7, pp. 307–325, 2001.

[42] N. Esfahani, K. Razavi, and S. Malek, "Dealing with uncertainty in early software architecture," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 21, ACM, 2012.

[43] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 163–186, 2012.

[44] T. Kelly, "Utility-directed allocation," in *First Workshop on Algorithms and Architectures for Self-Managing Systems*, vol. 20, 2003.

[45] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[46] S. Amundsen, K. Lund, F. Eliassen, and R. Staehli, "Qua: platform-managed qos for component architectures," in *Proceedings from Norwegian Informatics Conference (NIK)*, pp. 55–66, 2004. [retrieved: March, 2014].

[47] W. Watthayu and Y. Peng, "A bayesian network based framework for multi-criteria decision making," in *Proceedings of the 17th international conference on multiple criteria decision analysis*, pp. 6–11, Citeseer, 2004.

[48] C. C. Bennett and T. W. Doub, "Temporal modeling in clinical artificial intelligence, decision-making, and cognitive computing: Empirical exploration of practical challenges," in *Proceedings of the 3rd SIAM Workshop on Data Mining for Medicine and Healthcare (DMMH). Philadelphia, PA, USA*, 2014.

[49] T. H. Bui, M. Poel, A. Nijholt, and J. Zwiers, "A tractable hybrid ddn–pomdp approach to affective dialogue modeling for probabilistic frame-based dialogue systems," *Natural Language Engineering*, vol. 15, no. 02, pp. 273–307, 2009.

[50] P. C. Da Costa and D. M. Buede, "Dynamic decision making: a comparison of approaches," *Journal of Multi-Criteria Decision Analysis*, vol. 9, no. 6, pp. 243–262, 2000.

[51] T. H. Bui, M. Poel, A. Nijholt, and J. Zwiers, "A tractable hybrid ddnpomdp approach to affective dialogue modeling for probabilistic frame-based dialogue systems," *Natural Language Engineering*, vol. 15, pp. 273–307, 4 2009.

[52] A. R. Cassandra, *Exact and approximate algorithms for partially observable Markov decision processes*. Brown University, 1998.

[53] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra, "Solving pomdps by searching the space of finite policies," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 417–426, Morgan Kaufmann Publishers Inc., 1999.

[54] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs," *Philosophical Transactions (1683-1775)*, pp. 370–418, 1763.

[55] B. McCabe, S. M. AbouRizk, and R. Goebel, "Belief networks for construction performance diagnostics," *Journal of Computing in Civil Engineering*, vol. 12, no. 2, pp. 93–100, 1998.

[56] W. Dargie, "The role of probabilistic schemes in multisensor context-awareness," in *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pp. 27–32, IEEE, 2007.

[57] D. Bernoulli, "Exposition of a new theory on the measurement of risk," *Econometrica: Journal of the Econometric Society*, pp. 23–36, 1954.

[58] R. Schäfer and T. Weyrath, "Assessing temporally variable user properties with dynamic bayesian networks," in *User Modeling*, pp. 377–388, Springer, 1997.

[59] Z. W. Bhatti, N. Z. Naqvi, A. Ramakrishnan, D. Preuveneers, and Y. Berbers, "Learning distributed deployment and configuration trade-offs for context-aware applications in intelligent environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 6, no. 5, pp. 541–559, 2014.

[60] N. Z. Naqvi, D. Preuveneers, and Y. Berbers, "A quality-aware federated framework for smart mobile applications in the cloud," *Procedia Computer Science*, vol. 32, pp. 253–260, 2014.

[61] A. Ramakrishnan, S. N. Z. Naqvi, Z. W. Bhatti, D. Preuveneers, and Y. Berbers, "Learning deployment trade-offs for self-optimization of internet of things applications," in *Proceedings of the 10th International Conference on Autonomic Computing, ICAC 2013*, pp. 213–224, 2013.

[62] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[63] B. Houlding, *Sequential decision making with adaptive utility*. PhD thesis, Durham University, 2008.

Car Drive Classification and Context Recognition for Personalized Entertainment Preference Learning

Thomas Christian Stone
BMW Group
Munich, Germany
Email: thomas.stone@bmw.de

Stefan Haas
SAP SE
Munich, Germany
Email: stefan.haas02@sap.com

Sarah Breitenstein
BMW Group
Munich, Germany
Email: sarah.breitenstein@bmw.de

Kevin Wiesner
Institute for Informatics
LMU Munich
Munich, Germany
Email: kevin.wiesner@ifi.lmu.de

Bernhard Sick
Intelligent Embedded Systems
University of Kassel
Kassel, Germany
Email: bsick@uni-kassel.de

***Abstract*—The automotive domain, with its increasing number of comfort and infotainment functions, offers a field of opportunities for pervasive and context-aware personalization. This can range from simple recommendations up to fully automated systems, depending on the information available. In this respect, frequent trips of individual drivers provide promising and interesting features, on the basis of which, usage patterns may possibly be learned and automated. This automation of functions could increase safety as well as comfort, as the driver can concentrate more on the experience of driving instead of repeatedly and manually adjusting comfort- or entertainment-related systems. To identify frequent driving contexts in a set of recorded signal in a vehicle, e.g., GPS tracks, this paper presents two different clustering algorithms: First, a hierarchical *Drive-Clustering*, which combines drives based on their number of common GPS points. Second, a *Start-Stop-Clustering*, which combines trips with the same start- and stop-cluster utilizing density based clustering. The *Start-Stop-Clustering* showed particularly good results, as it does not depend on the concrete routes taken to a stop position and it is still able to detect more trip clusters. To predict these drives, a Bayesian network is presented and evaluated, with logged trip data of 21 drivers. The Bayes Net uses context information, i.e., the time, weekday and the number of people in the car, to predict the most likely drive context with high accuracy. A new automated entertainment source selection algorithm demonstrates the usefulness of the retrieved information. The algorithm learns and predicts a driver's preferences for selected entertainment sources depending on recognized drive contexts.**

Keywords–Context-aware Vehicle; Spatial Clustering; Drive Context Prediction; In-Car Infotainment; Automation

I. INTRODUCTION

Many different definitions for *context* exist, depending on the domain and conception. In [1], the frequent drives of a car owner considered *contextual information* useful for vehicle personalization.

In common literature, there seems to be a general notion for the meaning of the term *context*. However, up until now, there is no single definition accepted as the common standard. In [2], *context* is described as "... any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [3] claims to have found over a 150 definitions for *context*. Despite lacking a single, universally accepted definition, there is no argument about it's usefulness in certain applications. Context-awareness is considered an important building block in the development of intelligent systems as it is said to significantly improve the interaction between a user and a system. Knowledge about a specific context is normally gathered by sensor readings and their interpretation [4], [5]. In the course of this article, *context* will be considered any piece of information that can be aggregated in a vehicle and enables "intelligent behavior" of in-car systems. This includes not only information such as daytime, weekday, number of passengers, fuel level and frequent trip targets, but also the driver's control behavior in terms of the car's functions.

With its increasing number of comfort and infotainment functions, the automotive domain offers a unique field of opportunities for context-sensitive functions. In recent years, many different context-aware advanced driver assistance systems (ADAS) have been introduced. They depend on information provided by dedicated sensor systems, particularly in the areas of safety and comfort. The lane departure warning system (LDW), adaptive cruise control (ACC) and intelligent speed adaption (ISA) are well-known examples for context-aware ADAS.

Another interesting and promising context to advance vehicle personalization is the drive itself. Above all, the repeated

drives of a person offer a lot of potential for finding consistent usage patterns. Subsequently, the found user behavior can be used for automating certain comfort functions. For example, if a driver usually checks received emails on the way to work or likes to listen to the news, the vehicle could adapt to these preferences by recognizing the drive context as a regularly drive to work and by automating the desired functions. This automation of functions could improve safety as well as comfort because the driver is no longer forced to adjust his personal settings by himself.

In the following, we will describe and evaluate different methods for the detection and prediction of repeated drives of individual drivers. To develop and evaluate our proposed methods, we had the possibility of utilizing recorded vehicle sensor data of 21 drivers collected over several months by a data logger. The collected data includes many different sensor signals exchanged between the different in-car electronic control units (ECU) over the Controller Area Network (CAN) bus, ranging from Global Positioning System (GPS) position to seat belt status.

The contributions of our article are two novel clustering methods for detecting repeated trips of individual drivers, a novel distance measure based on the Jaccard distance for comparing GPS tracks and a hybrid Bayesian network for predicting frequent drive contexts right away from the start of the trip based on contextual information, e.g., the time of the day and the number of passengers in the car. The frequent drives and the additional context information will be used to infer the intention of drives, e.g., "drive to work" or "drive to spare time activity", what we consider as the *drive context*.

The article is structured as follows. Section II gives an overview on existing work in the fields of route prediction, route recognition, destination prediction, and place mining. Section III outlines two new spatial clustering methods for detecting the frequent drive contexts of a particular driver. In Section IV, we present a hybrid Bayesian network to predict the frequent drive contexts of an individual driver immediately from the start of the trip, or even during a trip. The results we obtained running the algorithms individually on the collected drive data of each driver are described in Section V. Additionally, in Section VI, we provide a case study for in-car infotainment automation based on the presented algorithms. The results prove our claim about the usefulness of the presented algorithm. We close our work in Section VII with a summary and an outlook on possible future work.

II. Related work

Route recognition and prediction systems have been proposed in many different works [6], [7], [8], [9], [10]. In the majority of these publications, the general way to predict, and respectively recognize, the current route is based on the comparison of the current driving trajectory to previously recorded trajectories through suitable distance measures. Comparing GPS tracks can not be done with classic L_p metrics due to their length related inequality, dimension and noise. As a result, more elastic similarity measures are necessary. Already proposed distance measures are, for instance, based on the longest common sub-sequence (LCSS) algorithm [6], [11], [12], the Hausdorff distance [7] or the Jaccard distance [13]. In [11], this simple instance based learning approach of comparing the current route to already recorded routes is further enhanced by the inclusion of contextual information, e.g., time of the day, to better differentiate overlapping routes.

Probabilistic approaches for route and destination prediction have been presented amongst others in [13], [14], [15] and [16]. The investigated prediction methods are frequently based on Bayesian techniques and include additional contextual information, such as the time of the day, the particular weekday or even background information about locations to infer the most likely route or destination [16]. By contrast, [15] uses an unspecified type of Markov model instead of a Bayesian approach to predict the next location of a user.

Identifying personally important places of users in recorded GPS data has, for example been investigated in [17], [18], [19], [15], [10] and [20]. Density based clustering hereby proved more efficient than classic partitioning algorithms like k-means [21], [22], [17], [18], as the final clusters only consist of dense regions in the data space. Regions of low object density are not included in the final clusters and are considered as noise.

Also, there has been work on using location based contextual information in proactive recommendation and automation. In [20], a probabilistic approach was presented for learning individual locations of interest. The learned locations were then used for recommendation and automation of a vehicular comfort function. The approach of learning an explicit user preference model proves helpful, especially for integration of user feedback and uncertainty quantification.

Our work differs from existing publications, as we focus on the personal, repeated drives of individual drivers and their prediction. This helps recognizing individual drive contexts. The drive contexts themselves denote regular drives, e.g., "drive from home to work" or "drive from work to home". We consider the drive contexts as the basis for learning a driver's control behavior of certain functions. Therefore, the learned behavior is useful for recommendation and automation, which we will prove in a short-term study.

III. Detecting frequent drives

The basis for drive context recognition will be the frequent drives of a driver. To detect frequent drive clusters of an individual driver, we present and evaluate two different spatial clustering methods explained in the following two subsections. *Drive-Clustering* is based on the Jaccard distance and compares whole trajectories using hierarchical clustering. In contrast, *Start-Stop-Clustering* focuses on more semantic similarity measurement of routes, based on the determination of frequent start and stop positions of a particular driver. The goal of both algorithms is to identify repeated patterns in the set of recorded GPS tracks in order to detect repeatedly occurring drive contexts, e.g., drives from home to work. In Section V, we compare the obtained results of both algorithms applied to our test data set.

A. Drive-Clustering

An important factor in cluster analysis is a similarity measure to determine the distances between elements contained in the data, for the purpose of grouping similar elements together in clusters. In trajectory data the standard way for identifying patterns is to compare whole trajectories. In our case, the trajectory data of each drive is stored as a sequence of GPS points $S_i = \{p_{i,1}, p_{i,2}, ..., p_{i,n}\}$, with $p_{i,1}$ being the start point of the drive and $p_{i,n}$ being the end or stop point.

To compare two point sequences we use a dissimilarity

measure based on the well known Jaccard distance, which measures dissimilarity between sample sets [23] (see equation (1)):

$$d(X,Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}. \quad (1)$$

Our dissimilarity measure thereby calculates the intersection of the two GPS sequences S_i and S_j by counting the number of common points $NOCP(S_i, S_j)$ contained in both sequences starting from the shorter sequence (see equation (2)). This number of common points is then divided by the number of points contained in the shorter sequence $min(|S_i|, |S_j|)$. In order to obtain a dissimilarity measure the quotient is subtracted from 1, so that a result of 0 indicates maximum similarity and a value of 1 maximum dissimilarity.

$$d(S_i, S_j) = 1 - \frac{NOCP(S_i, S_j)}{min(|S_i|, |S_j|)}. \quad (2)$$

GPS points of two geometrically similar trajectories are unlikely to have exactly the same coordinates. Even if a drive can be done exactly the same way several times, the GPS sequences will not be equal because of the noisy nature of GPS measurements. Hence, it is necessary to define a threshold distance Θ, e.g., 50 meters, to decide whether two GPS points from two sequences can be considered as "equal". This is necessary to find the GPS points shared by both sequences, i.e., the *"common points"*. The threshold needs to be defined dependent on the logging frequency, assumed the GPS measurements were done periodically. In our case, the logging frequency is $f = 1Hz$. If we, for example, consider 135 km/h as the maximum vehicle speed, the maximum distance between two subsequent measured GPS points will be $(135 \cdot 1000)m/3600s = 37.5m$. In the evaluation we set the threshold to 50 meters, which is sufficient for driving speeds up to 180 km/h with a logging frequency of $f = 1Hz$.

The number of common points (NOCP) algorithm iterates over all points $p_{i,k} \in S_i$ included in the shorter sequence and tries to find at least one point in the other sequence $p_{j,l} \in S_j$ whose distance is less or equal than the defined threshold distance Θ. If the set of found points in range is not empty, the number of common points counter is increased. Consequently, the presented distance measure is more elastic than distance measures based on dynamic programming, such as the longest common sub-sequence (LCSS) or dynamic time warping (DTW), as it is able to match several elements of one sequence to just one element of the other sequence, without taking into account the sequence ordering. This behavior is important in our case to handle traffic jams and different driving speeds. The implementation of the number of common points (NOCP) function can be significantly sped up by storing the queried sequences' points in a *k-d tree* [24].

To calculate the distance between two-dimensional GPS points we use a simplification of the *haversine* formula [25] based on the Euclidean distance, which in contrast to the standard Euclidean distance allows metric parametrization of our algorithms (ϕ latitude, λ longitude) (see equation (3)).

$$dist(\phi_1, \lambda_1, \phi_2, \lambda_2) = (((111.3 \cdot \cos(\frac{\phi_1 + \phi_2}{2}) \cdot (\lambda_1 - \lambda_2)^2) + (111.3 \cdot (\phi_1 - \phi_2)^2))^{\frac{1}{2}} \cdot 1000. \quad (3)$$

The *haversine* formula calculates the distance of two points on a sphere along their respective great-circle. While the original *haversine* formula is costly to calculate with all it's trigonometric functions, the given approximation is fast and precise for world GPS coordinates.

In order to avoid the problem of a much shorter sequence being contained in a longer sequence and to speed up the comparison, the number of common points in the two sequences is only calculated, when the first and last points of the two sequences are sufficiently similar. This means their respective distances do not exceed a predefined threshold, e.g., 250 meters ($p_{i,1} \sim p_{j,1}$ and $p_{i,n} \sim p_{j,m}$). Otherwise, the maximum dissimilarity value 1 is returned without any further calculation (see equation (4)).

$$d_{opt}(S_i, S_j) = \begin{cases} 1 - \frac{NOCP(S_i,S_j)}{min(S_i,S_j)}, & \text{if } p_{i,1} \sim p_{j,1} \\ & \text{and } p_{i,n} \sim p_{j,m} \\ 1, & \text{otherwise} \end{cases} \quad (4)$$

To group similar routes in clusters, we use *single-linkage clustering*, an *agglomerative hierarchical clustering method*, starting from single GPS sequences. We use the function d_{opt} for distance measurement. A merging threshold ε decides whether two clusters are close or similar enough to be merged, e.g., $\varepsilon = 0.05$. The clustering stops when there are no clusters left for merging. The smaller the value ε the more similar the trips contained in a cluster are, but in general, less clusters will be merged. This threshold will cut the *dendrogram* at a certain level and lead to the final drive clusters. The resulting clusters without enough observations can be considered *outliers* and will be deleted. To predefine the minimum cluster size we use another parameter $MinDrives$. As every point in our clusters represents a single drive, $MinDrives$ represents the parameter $MinPoints$ introduced in the density based clustering in [22]. This renaming was done for the purpose of convenience.

B. Start-Stop-Clustering

Another way of determining frequent drives of a certain driver is based on his frequent start and stop positions of drives. The start and stop positions are the GPS locations, where the car is started and parked respectively. In contrast to the above presented trajectory clustering method, this method focuses on drives with the same start and stop positions, not on geometrically similar routes.

As the vehicle is typically not parked at the exact same coordinates, it is necessary to merge similar parking positions to *start-stop-clusters*. To obtain these frequent start and stop position clusters of a particular driver, we use a density based clustering, the DJ-Cluster algorithm presented in [17], which is a simplification of DBSCAN [22], [26]. Density based clustering has the advantage of explicitly eliminating outlier points compared to partitioning clustering, e.g., k-means [21], [26]. As we are only interested in dense regions included in the set of start and stop positions of an individual driver in order to identify frequent drive contexts, density based clustering is suitable for our task.

Consequently, the first step in Start-Stop-Clustering is to calculate dense regions of start and stop positions in the set of GPS sequences and to store the cluster IDs of every GPS sequences' start and stop points. Therefore, it is necessary to specify the two parameters $MinDrives$ and ε, representing the minimum cluster size and search radius respectively. Figure

1 shows an example of a dense point cluster found in the drive data of a particular driver with $\varepsilon = 100$m.

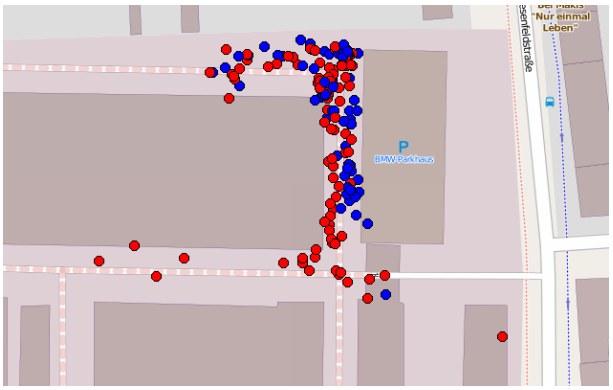


Figure 1. Visualization of the start (red) and stop points (blue) of a driver. All shown points are included in the same point cluster.

The equality of GPS sequences for Start-Stop-Clustering is determined by a binary function (see equation (5)).

$$d(S_i, S_j) = \begin{cases} 0, & \text{if } C_s(p_{i,1}) = C_s(p_{j,1}) \\ & \text{and } C_e(p_{i,n}) = C_e(p_{j,m}) \ . \\ 1, & \text{otherwise} \end{cases} \quad (5)$$

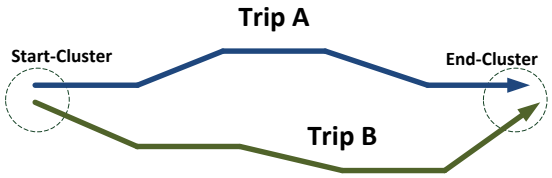


Figure 2. Illustration of a route-independent *Start-Stop-Cluster*.

Two GPS sequences S_i and S_j are considered as equal, when their corresponding start ($p_{i,1}$, $p_{j,1}$) and stop points ($p_{i,n}$, $p_{j,m}$) lie in the same start C_s, respectively end cluster C_e. Hence, the final frequent drive clusters are comprised of GPS sequences whose start and stop points lie in the same dense region or point cluster and therefore have the same cluster IDs. The clusters are direction-dependent just like those obtained with the above presented Drive-Clustering approach. However, the drives included in a *Start-Stop-Clustering* drive context cluster do not necessarily follow the same routes (see Figure 2). To predefine the minimum cluster size we also use the $MinDrives$ parameter.

IV. Predicting Frequent Drive Contexts

The frequent drive clusters will be merged with additional information, forming a new context we will call *frequent drive context*. A *drive context* denotes a contextual description, or at least a semantic clustering of drives, depending on the intention of the drive, e.g., "drive to work" or "drive to gas station". *Frequent drive contexts* denote drives that happen to be periodic or frequent in a sense, e.g., daily drive to work and back home. As we will concentrate on frequent drives from now on, we will refer to them simply as *drive contexts*.

To predict frequent drive contexts that have been identified with one of the above presented methods, we propose a hybrid Bayesian network, incorporating more than just the clustered frequent drives as location based features. This is a basic requirement stated in [27], where contexts should not only consist of location based features. The structure of the network is shown in Figure 3.

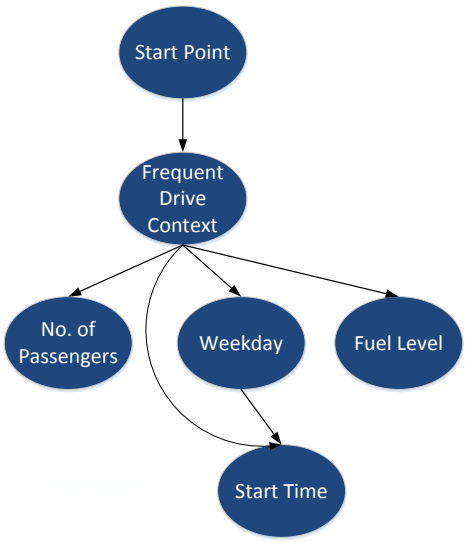


Figure 3. Topology of the hybrid Bayesian network for predicting the most likely frequent drive context.

Using the start point of the drive we are able to eliminate impossible contexts, e.g., a drive from work to home if the start point is home, which significantly reduces the possible contexts, prevents false positives and speeds up the implementation. The variable *Frequent Drive/Context* represents the *a priori* probability distribution over the set of identified drive contexts, already constrained by the current start point. The variables *Weekday*, *No. of Passengers* and *Fuel level* are conditionally independent of each other given the *class* variable *Frequent Drive Cluster*. The variables described so far all underlie a discrete probability distribution. The fuel level is discretized to "at least half full" and "not half full", in order to ease modeling. The number of passengers are discretized to $1, 2, 3, 4, 5$.

In contrast to the other probability variables, we model the variable *Start Time* as a continuous variable. By the edges between *Frequent Drive/Context*, *Day* and *Start Time* we receive a drive context dependent start time *probability density function* (PDF) for every single day. This enables a stronger differentiation between the drive contexts, as the start time probabilities for the different contexts are also dependent on the day.

To approximate the probability density function for the start times associated with a certain drive context we use *kernel density estimation* (KDE) (equation (6)) with a Gaussian kernel (equation (7)) and Scott's *rule of thumb* (equation (8)) for bandwidth selection h [28]:

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h}). \quad (6)$$

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right). \quad (7)$$

$$h_{scott} = n^{-1/(d+4)}. \quad (8)$$

n is the number of observed starting times for approximating the underlying probability distribution. $K \in [0, 24] \rightarrow \mathbb{R}_0^+$ defines the kernel function used in the estimator, with $x \in [0, 24]$ being the daytime and d the dimensionality of the problem.

By using *kernel density estimation*, we receive a continuous approximation of the probability density of starting points (see Figure 4).

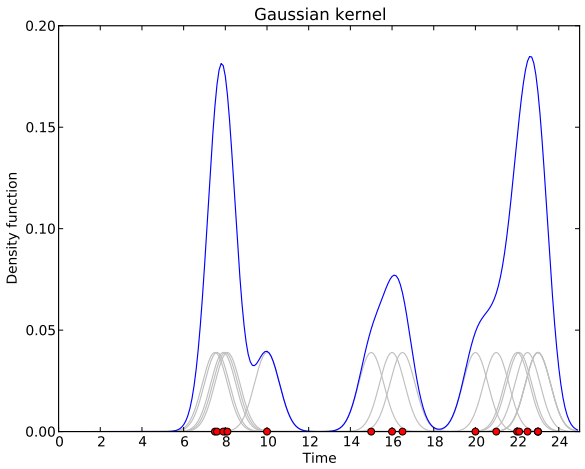


Figure 4. Example of a probability density function for the *Start Time* variable of a particular drive context.

When a drive context has not occurred before, at a certain day or time, the probability for the whole context will be zero. This kind of behavior is not always acceptable. A *Laplacian correction*, also called *Laplacian estimator*, is a common technique to solve this problem, as it adds observations to the dataset for unseen entities to prevent zero probabilities. We deliberately do not use *Laplacian correction*, thereby improving on false positives. In general, we assume a long-term observation phase for our proposed system, effectively eliminating this problem.

The probability for a certain context C, given the start point s, the weekday d, the time t, the number of persons in the car p and the fuel level f, can then be calculated with the following formula:

$$P(C|s,d,t,p,f) \propto P(C|s)P(d|C)P(t|d,C)P(p|C)P(f|C). \quad (9)$$

The context C_i leading to the highest probability value $P(C_i|s,d,t,p,f)$ is then assumed to be the present context:

$$\arg\max_{C_i}\{P(C_i|s,d,t,p,f)\}. \quad (10)$$

The prediction in the presented form lacks the possibility to make online adaptions and corrections of the predictions. This may be necessary due to ambiguous information at the beginning of a trip, which prevents a good prediction of the current drive context. In the case where the starting points do not deliver enough information to predict the frequent drive cluster, the Bayes Net can be re-evaluated when the drive cluster is known during the trip. For instance, this can be achieved by constantly calculating the similarity of the current route to the frequent drive clusters. To that end, the similarity measure from Section III-A can be used. If the predicted frequent drive class of the current drive changes, the presented Bayes Net re-predicts the current drive context.

V. Evaluation

To evaluate the described methods, we have access to a data set collected from 21 drivers over several months. The logger used for collecting the data, records all kinds of data bus traffic, also when the car is not moved, e.g., when the electronic key is pressed. To filter out this unwanted "noise", we only used recorded data for our evaluation where the vehicle was at least moved 1 kilometer (air-line distance). The number of filtered drives per driver ranged from 225 to 983 drives. This wide-spread distribution is related to the 3 to 8 months of recording duration and the individual use of the cars. The majority of the subjects ranged between 400 to 600 recorded drives.

A. Drive clustering

Figures 5 and 6 show the results obtained applying Start-Stop-Clustering and Drive-Clustering to the data set. Figure 5 illustrates the average number of found clusters for different minimum cluster sizes ($MinDrives = \{3, 5, 10\}$). Figure 6 presents the average share of frequent drives of the total quantity of drives, i.e., "frequent" and "non-frequent" drives. As no ground truth could be gathered, the analysis must follow qualitative and quantitative reasoning.

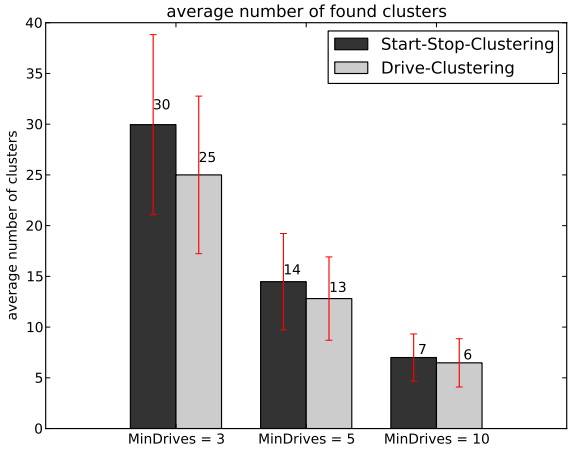


Figure 5. Average number of found clusters with Start-Stop- and Drive-Clustering dependent on the minimum number of drives contained in the clusters ($MinDrives$).

As one can see, Start-Stop-Clustering is on an average able to identify more clusters than Drive-Clustering (see Figure 5). However, with increasing minimum cluster size, the difference between the average number of found clusters by Start-Stop-Clustering and Drive-Clustering decreases. This leads to the

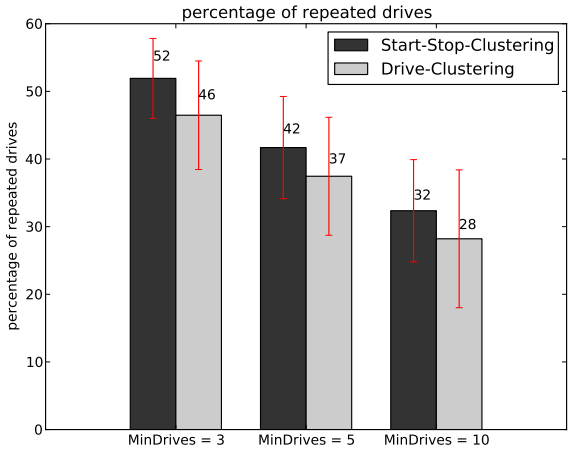


Figure 6. Percentage of repeated drives identified with Start-Stop- and Drive-Clustering dependent on the minimum number of drives contained in the clusters ($MinDrives$).

assumption that for frequent drives ($MinDrives = 10$), drivers usually have a preferred route that they normally take, whereas for less frequent drives ($MinDrives = 3$) they also take different routes to the same destination. Furthermore, Start-Stop-Clustering includes all route alternatives (see Figure 6), thus, assigning a larger fraction of the overall number of drives to a repeated drive cluster.

As we are rather interested in detecting frequent drive contexts than the frequent routes taken by a driver, Start-Stop-Clustering is more appropriate for our use case. Large clusters ($MinDrives \geq 10$) may provide promising and interesting contexts, on the basis of which usage patterns may possibly be learned and automated. The average fraction of trips, repeated at least 10 times by the participants during the survey, amounts to approximately 30% of the overall trips (see Figure 6).

To keep the set of frequent driving contexts up-to-date one could use a shifting time frame and only consider drives for the cluster calculation that for example occurred during the last 6 months. This would lead to a slow exclusion of no longer appearing driving contexts over time and also limit the amount of data used for the context identification.

B. Prediction

To evaluate our proposed Bayesian inference system for predicting frequent drive contexts, we made use of cross-validation and focused on clusters identified by Start-Stop-Clustering with a cluster size larger than 10 drives. The cross-validation was done for every driver to even out the different recording times and to be able to differentiate between different types of drivers. The applied evaluation method was leave-one-out cross-validation to cope with the small data sets.

Figure 7 shows the overall prediction result for all drives, including also non-frequent drives, as well as the prediction result for solely frequent drives belonging to a cluster. The prediction result improves significantly, to almost 100% (∼97%), when a prediction result is considered correct when lying within the top 3 predictions.

Evaluating the top 3 results shows the usability in recommendation systems. Presenting the user a recommendation for each of the n most likely drive contexts is a common setup. In the case of recommendation based on learned user preferences, showing the most likely recommendation to the most likely contexts leads to a higher chance of user acceptance.

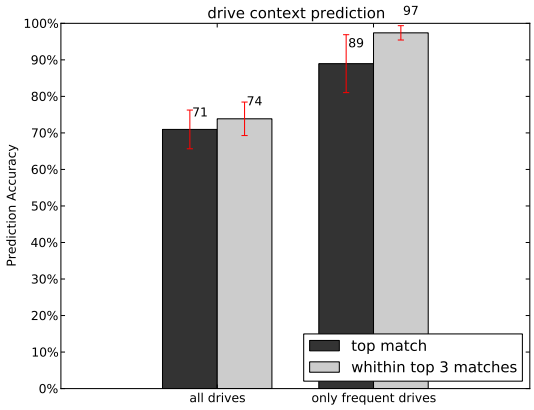


Figure 7. Prediction result for all drives and only frequent drive contexts ($MinDrives = 10$).

The differentiation between the different drive contexts is relatively accurate (∼ 89% respectively ∼97% for top 3 matches). Moreover, in Figure 8 one can see that, when considering all drives, the main share in false predictions not lying within the top 3 matches is produced by false positives. A false positive is the classification of a starting drive as a "frequent drive", when it is not. A large fraction of false positives could be detected correctly (∼60%), but as there might be highly frequent start and stop positions, e.g., home coordinates, with overlapping context information, e.g., time and weekday, some infrequent drives were predicted as belonging to a frequent drive context.

In the evaluation, we used a binary probability distribution for the day variable (workday, weekend) due to the relatively small minimal cluster size of 10 drives. It might be possible to achieve a better recognition of infrequent drives by assuming a discrete probability distribution for every day (Monday, Tuesday, Wednesday, etc.), which would also lead to time probabilities for every day for each drive context. However, this would only make sense with a higher minimal cluster size, in order to get representative probability distributions for every day.

Compared to the rate of false positives the rate of true negatives is extremely low and underlines the accuracy of our inference system related to the prediction of frequent drive contexts (see Figure 8). However, eliminating false positives is crucial in order to not annoy the driver with unwanted function automation and might only be solvable with little driver interaction. A solution could be to provide the driver with the top 3 most likely contexts. Then, the driver is able to choose the most appropriate one. If none is selected by the driver after a certain driving time, the system assumes that, in the current situation, no function automation is wanted by the driver.

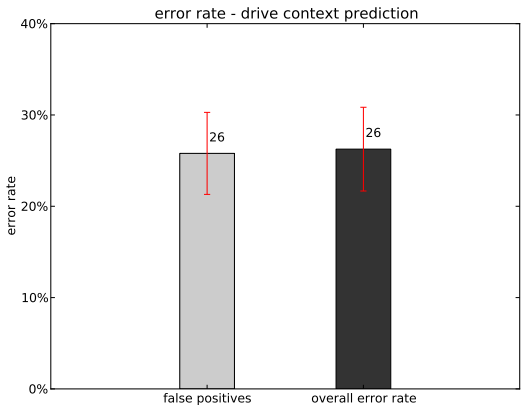


Figure 8. Overall prediction error rate and the share of false positives at the overall error rate.

VI. Context-aware automation case study

The use of comfort and infotainment functions in a car generally depends on the driver's preferences. In turn, those depend strongly on the intention for the drive, represented by *drive contexts*. As a consequence, the use of comfort functions depends on the drive context, e.g., *commuting drive* or *regular fitness club visit*. The class of climatic functions are a good example. For instance, while a driver heats up the car in the morning, lower temperatures may be wished-for after regular fitness center visits.

In the case of infotainment functions, e.g., integrated TV or audio player, this dependence is also visible. The selected audio source can be related to the driver's mood, time of the day, or again on the drive context. A driver may always listen to a certain radio station for traffic information on the way to work. But then, leaving work, she listens to CDs, music from the connected smartphone, or any other device connected. While this can have different reasons, the trip's goal or the intention behind the trip may be the most important. This leads to the idea of using the previously recognized drive contexts for audio source automation.

Starting from this idea and the presented drive context recognition, an explicit user preference model for infotainment functions can be realized. In the following, we will present an audio source selection automation algorithm. The algorithm will serve as a showcase for the usefulness of the drive context information in infotainment automation. Also, it will demonstrate a modular, purely probabilistic view for the use of an explicitly modeled user preference relying on contextual information.

A. Entertainment source selection

In a modern car, there are several different audio sources to choose from. Typically, these include various types of radio sources, TV, internal storage and connected personal devices or a subset of these. The driver can select one of them at a time or disable all audio sources. Depending on the car's user interface, disabling audio is either done by disabling audio output or decreasing volume to zero.

The goal of the following case study is to showcase the use of the drive context recognition integrated into an automation of an infotainment function. The chosen function is the selection of the current entertainment source in the car. This means that we would like to recognize a driver's behavior in terms of selecting entertainment sources depending on the current drive context.

For the justification of declaring the *drive context* as the main context for entertainment source selection, we conducted a preliminary study. 30 subjects were interviewed and questioned about their user behavior in terms of infotainment systems. This also included the use of the navigation system, eliminating the need for the prediction of the current drive cluster related to the current drive context.

While 80% of the subjects always use the navigation system, 7% use it frequently and 13% use it occasionally, no one would never use it. This indicates the necessity of the prediction of the current drive cluster, hence the target of the current drive.

28 of the 30 use more than just one entertainment source. While two subjects would listen to the radio only, one of them depicted music as generally not important. The different sources used are listed in Figure 9.

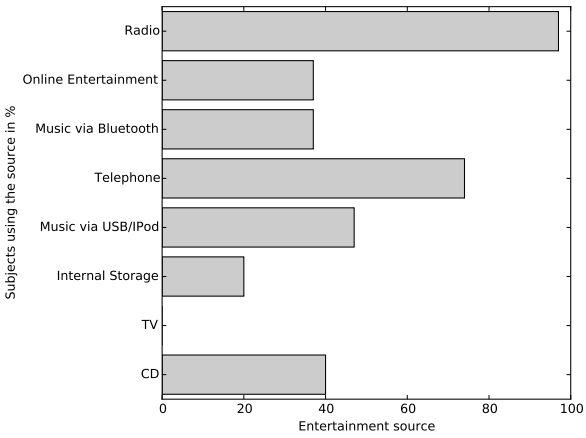


Figure 9. The amount of subjects using the different entertainment sources available.

25 of the 30 subjects stated to choose the entertainment source depending on the *drive context*. One answer was given, that because of the spare time drives mostly going abroad, the subject listened to the online entertainment rather than listening to the local radio. Another representative answer was that the subject listens to the radio to get to work and "moderately" start into the day, while listening to CDs or making calls on the way back home.

Three of the 30 subjects would never change their behavior according to the co-driver or the passengers in the back. The rest of the subjects would either give the control of the entertainment source to the passengers or completely go without any entertainment source. In Figure 10, additional influencing contexts were given.

The results from the interviews give some important indications about using the *drive context* for learning the driver's entertainment source behavior. The contextual information used in Section IV, daytime, number of passengers and the frequent drive clusters are the most important information for predicting the selected entertainment source. Therefore,

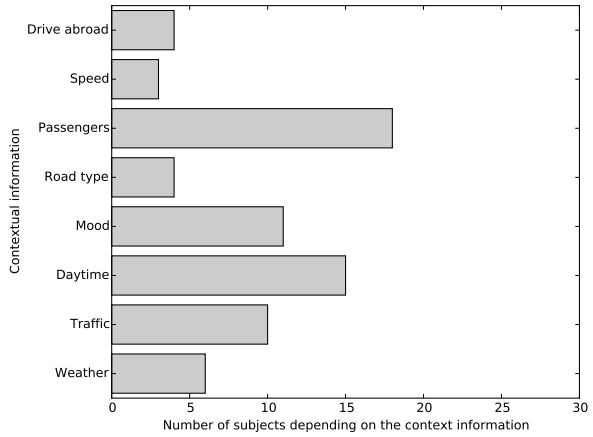


Figure 10. Additional dependencies influencing entertainment source selection.

the *drive clusters* abstraction delivers most of the information necessary for predicting the selected sources.

B. Probabilistic view

This case study is targeted at automating a comfort function through imitating the driver's control behavior. In [20], a similar problem is formulated: Proactive recommendation or automatic activation of a certain camera-based comfort system at locations of interest. The locations are learned by observing a driver's individual use of the camera system. As an outcome, a modular system for location based activation of comfort functions is presented. It relies on a probabilistic view on the integration of abstract contextual information into the process of automation. This has several advantages over non-probabilistic methods, also discussed thoroughly in [20]. The main goal in [20] is formulated as

$$p(A|B) = \sum_{O} p(A, O|B) \tag{11}$$

being the probability of the driver intending to activate a comfort function under observation B and learned locations O. With some basic assumptions, the probability of an intended activation A can be simplified to

$$p(A|B) \approx p(A|O_j) \cdot p(O_j|B), \tag{12}$$

where O_j is a context, describing an "abstract location". The observation B can be any information currently accessible to the car. This separation of activation and location context in [20], makes it possible to model both contexts differently. In [20], this was used to implement user feedback and uncertainty quantification for better decision making.

This approach of separating the intention of an activation and the major influencing contexts of the decision can be adapted to our showcase. Instead of the probability of an activation A, we want the probability about the possibly selected audio sources. The selections of available audio sources will be denoted by $E = (E_n)_{n \in N}$, with $N = \{0, 1, \ldots, \sharp\text{sources} - 1\}$. Every E_n represents a different audio source, e.g., radio, DVB-T or disc player. E_0 is not an active audio source itself, but implies *"disabled audio"* output of the entertainment system. This simple definition will be useful later on, enforcing

$$\sum_{i \in N} p(E_i) = 1. \tag{13}$$

This implies that we always want an answer for the automation mechanism.

The targeted automation does not depend on the notion of a learned location of interest, but rather on the current drive context. Therefore, the probability of an audio source E_i being selected under individually learned drive contexts can be described as

$$p(E_i|B) = \sum_{C} p(E_i, C|B) \tag{14}$$

$$= \sum_{C} p(E_i|C, B) \cdot p(C|B). \tag{15}$$

$C = (C_m)_{m \in M}$ is the family of observable drive contexts indexed by $M = \{0, 1, \ldots, \sharp\text{drive contexts} - 1\}$.

This is analogous to the situation in [20]. It is important to notice, that in [20], the probability of "no activation desired" is not calculated explicitly, but rather included in working with the probability of a wanted activation. In our case of automating the entertainment selection, the *"disabled audio"* selection is treated as another selection. Thus, the system always selects and activates a source.

In [20], some assumptions were made regarding the probability densities involved in equation (11). This allows the deduction of equation (12) for estimating $p(A|B)$. In our case study, similar assumptions can be made to estimate $p(E_i|B)$:

1) Given the information of the current drive context, any other information denoted by B does not gain additional valuable information for the automation. This is inferred from our basic assumption that the drive context is the major dependence for the driver's audio source preference. This means all information of the observation B is included in the context. This assumption induces the following simplification:

$$p(E_i|C, B) \approx p(E_i|C) \tag{16}$$

In the case of having more than one context, this can be a dangerous assumption, but is viable for the case study.

2) The previous evaluation of the presented drive context recognition in Section V-B shows a high accuracy. Taking the k most likely drive contexts into account, the accuracy is close to 100%. If I is the index set for the k most likely selected audio sources,

$$\forall j \notin I : p(C_j|B) \approx 0 \tag{17}$$

can be seen as a viable assumption. Taking an even sharper condition, setting $k = 1$, the same assumption as in [20] can be made, leading to

$$\forall i \notin N \backslash I : \quad p(C_i|B) \approx 0 \tag{18}$$

$$i \in I : \quad p(C_j|B) \approx 1 \tag{19}$$

for C_j being the most likely drive context to be predicted. This approximation is supported by the high accuracy evaluated for only taking the most likely drive context.

Essentially, this simplifies the prediction of the audio source selection. Analogous to [20], the estimation of the probability for the selection of a specific source E_i simplifies to

$$p(E_i|B) \approx p(E_i|C_j) \cdot p(C_j|B) \tag{20}$$

In the case of a recommendation system, the system may be free to offer the driver to play the audio source with the highest likelihood. If a source would not be available, the system should not recommend it and, therefore, does not have to recommend anything at all. But, in the case of a fully automated system, logic dictates to choose and play the most likely audio source available. If a source is unavailable, the source with the closest likelihood will be selected. The automatically selected source E_{i_0} then is determined by

$$i_0 = \arg\max_{i \in I_{\text{avail}}} p(E_i|C_j) \cdot p(C_j|B) \tag{21}$$

where I_{avail} is the index set for the available audio sources. As $p(C_j|B)$ is the probability of the current drive context, its calculation can be seen in Section V-B. From now on we will call it *Drive Context Distribution*. The probability distribution of the selected source, i.e., $p(E_i|C_j)$ will be shown in Section VI-C. The term will be called the *Source Distribution*.

Still, there is another simplification possible in equation (21). Since the $p(C_j|B)$ is a constant in equation (21), it can be eliminated, not changing the decision process on E_{i_0}. Thus, under given context drive C_j, the automated selection of the audio source can be formulated as

$$i_0 = \arg\max_{i \in I_{\text{avail}}} p(E_i|C_j) \tag{22}$$

When radio is selected as an audio channel, the preferred radio channel should also be automated depending on the drive context. Therefore, we define all radio channels to be denoted by $R = (R_k)_{k \in K}$ with K being the index set for every listened or known radio channel. Analogue to the entertainment source selection, the most likely selected radio channel R_{k_0} given a specified drive context C_j can be formulated as

$$k_0 = \arg\max_{k} \arg p(R_k|C_j, E_1) \tag{23}$$

when E_1 denotes the radio source.

C. Selected source

For the final decision, which source must be selected by the presented system, *Source Distribution* $p(E_i|C_j)$ must be learned. This can be done in several ways, involving to "observe" the active source on a trip. We will present two intuitive approaches to decide which entertainment source is considered to be *"listened to"*. One will serve as example for a whole class of point evaluation methods, while the other is motivated differently. As the same ideas for finding the active entertainment source apply to the active radio source, it will be referred to as *active source of a trip* from now on. These two basic ideas are illustrated in Figure 11. Figure 11a illustrates the time-lines for three different trips from a drive cluster. The colors orange and blue imply the relative time of one of the two active sources along each trips time line.

The first approach is to observe the activated source at a predefined point in time, relative to the drive's beginning or ending and declare it the *active source of the trip*. In Figure 11b, the first source that is observed as being active at the beginning of the trip is declared to be the *active source of the trip*. Defining a point in time, to designate the currently selected entertainment source as the *active source of the trip* is difficult. While the targeted automation should work at the beginning of the trip, it is hardly a good idea to do so. Most automotive audio systems will start when starting the car and choose to play the last active source or radio station. If the driver now has a different preference, the automation system must recognize this. Defining when the driver has settled for an active source is also non-trivial, because the driver may change the source periodically. This is illustrated in the middle trip in Figure 11a.

The second approach is to decide the *active source* depending on the source switching behavior from the driver. Periodic changes of the *active source* may strongly depend on other influences from the environment and the content of the audio source itself. In Figure 11a, the top trip shows some switching between the two sources. While the orange source may be solely for entertainment purposes, the second one may be preferred by the driver as news or traffic information source. The driver then would switch in between sources when this information is wanted, e.g., when stuck in traffic. Taking the longest active source is acceptable, as the driver listened to it for the longest, thus the audio source delivering most of the preferred content to the driver.

As for the sake of practicability of this case study, the latter approach was used. The focus is on the demonstration of the drive context for entertainment source selection.

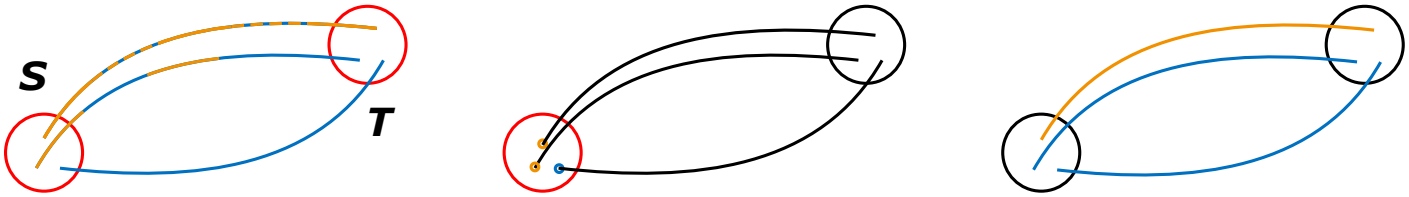
D. Case study data

For this case study of drive context for personalized entertainment source prediction, eight drivers participated in a short-term study. The drivers were provided with prepared cars, logging the standard bus systems and the central entertainment system. The logging system was a prototype explicitly developed for this study and had to be installed in the cars with connections to the internal data bus system of the central entertainment system. The cars were provided for about four to six days to each participating subject, being enough time to recognize the working time. This provided the data for an offline evaluation of the algorithms. The entertainment system could not be controlled externally, making testing the automation online impossible.

The drivers were also interviewed, allowing the comparison of the offline prediction and the subjects statements. This would ensure a higher expressiveness of the short-term results of the study.

Gathering enough data from the subjects over this short time span was not possible with every subject. For the previously presented drive context algorithm to work, at least several drives for every context must be observed. The $MinDrives$, declared in Section III-A, was set to $MinDrives = 3$ for the GPS route clustering, as it was proved to work accurately with small values for $MinDrives$ (see Section V). Also, the predicting algorithm for the source selection automation needs some observations of every drive context.

In the case of this study, only six subjects delivered enough data for a significant analysis. This is enough to give a coarse



(a) Two sources on three routes (b) Active source at the beginning (c) Longest active sources per route

Figure 11. Two approaches for tagging the *active source of a trip*: 11a illustrates an example for a drive context's current active source (orange and blue) on time scales from **S**tart to **T**arget. 11b and 11c demonstrate declaring either one of the listened sources or the longest listened source of a trip as the trip's observed active source.

quality indication for the usefulness of the drive context in automation and the integration for the entertainment source prediction.

E. Evaluation

For the evaluation of the in-field case study, the logs of the 6 subjects were analyzed. The analysis includes the clustering of frequent drives and subsequent recognition of the drive context. On top of the recognized context, the presented algorithm from Section VI-B was used to learn the subjects' current audio listening preference. The outcome of the comparison of the predicted information and the subjects' interviews are listed in Table I.

In the interview, the subjects were asked about their personal preferences of infotainment use. This included the dependence on the frequent drives, passengers, co-passengers and mood. The given information was useful for a better understanding of the preferences learned in the study. The information of the preferred audio sources itself was divided into three different categories of drive context: *home to work*, *work to home* and *non-work related / leisure* drives.

Table I was structured for ease of comparison between the prediction and the data from the interviews. While on the leftmost side are the numbers of 6 useful subjects, on the right side are the most three categories of detected drive contexts. Per category of drive context, the table includes the prediction of the presented automation algorithm, as well as the answer from the interview. The table shows the information given in the interview, while on the right, the algorithm estimated the user's preference.

A general problem was that the non-work related drives were not properly detected. This is due to the short-term of the study, as well as the time of the recordings. The cars were provided mostly over working days, while most non-work related drives are done at weekends. Nevertheless, Table I, showing the detected preferences and the information given at the interviews, indicates promising results.

The trivial case of a constant preference, non-dependent on drive context or any contextual information must not be a problem for the algorithm from Section VI-B. Subject number 4 never changes the active audio source and, therefore, is a good example for trivial preferences. The presented algorithm indeed has no problem recognizing the constant behavior and predicts radio and the channel properly and verifiable.

Subject 3 also shows the working of the preference recognition. The subject said it would not listen to different radio channels, thus, recognizing the radio as the preferred source was the primary target. The presented recognition always chooses the most likely radio channel as explained in Section VI-C. To stabilize the automation in a long-term study, the presented learning of entertainment source preferences has to implement a form of uncertainty quantification as shown in [20]. Also, the preference recognition does not take into account the use the telephone, because calls are not necessarily made for the same reason as audio source are selected. Therefore, *CD* was the best prediction result possible on work to home drives.

Subject 5 provided 4 days of logged information, mostly on workdays. Despite the small amount of data gathered, the prediction showed a significant difference in preference on work travel. While listening to CDs in the morning work travel, listening to the radio driving back home was clear and coincided with the data given at the interview. The non-work drives did not deliver enough information for a significant statement, but gave a coarse direction. The given influencing factors confirmed the use of the indicated information in Figure 10.

The presented case study showed that the presented drive context recognition works well as a basis for an explicit user preference model. Using the number of passengers and daytime information in the drive contexts is clearly a benefit as shown in the evaluation of the study. The influencing factors given at the interviews approve the benefits. This confirms the initial assumption, that the drive context is a major dependency of driver preferences in comfort functions and useful for recommendation and automation systems.

VII. Conclusion and Future Work

In this article, we investigated the detection and prediction of frequent drive contexts as an important building block for automatized vehicle personalization. We proposed two different spatial clustering approaches for identifying frequent drive patterns in a GPS data set. The route independent Start-Stop-Clustering is promising, as it detects patterns independently of the chosen routes. The presented Bayesian Net's accuracy in differentiating frequent drive contexts was about 89% respectively 97% for a top 3 match.

We also presented a case study incorporating the recognized drive context. The study showcased the usefulness of the presented drive context recognition algorithm, when

TABLE I. Offline evaluation example and interview comparison.

Subject	Home to work		Work to home		Non-work related		Influencing factors
	Driver	Predicted	Driver	Predicted	Driver	Predicted	
1	radio	DAB, $R_{1,0}$	radio	DAB, $R_{1,1}$	Bluetooth	-	daytime, co-driver
2	radio	DAB, $R_{2,0}$	radio/telephone	AuxIn	mostly radio	DAB, $R_{2,0}$	daytime, co-driver
3	radio	FM, $R_{3,0}$	CD/telephone	CD	CD/radio	radio	daytime, mood, co-driver
4	radio	FM, $R_{4,0}$	radio	FM, $R_{4,0}$	radio	FM, $R_{4,0}$	passengers
5	CD	CD	DAB, $R_{5,0}$	DAB, $R_{5,1}$	misc.	-	daytime, co-driver
6	radio	FM, $R_{6,0}$	radio/telephone	FM, $R_{6,1}$	radio	FM, $R_{6,2}$	weather, road type, passengers

learning driver preferences. The targeted comfort function was the entertainment source selection and proved to work very well. It also showed a modular way incorporating such context information. The next step should be a long-term study for significant confirmation of the results. Also, the number of contexts should be increased and techniques for reducing the burn-in phase of the preference learning should be investigated. This would include techniques ranging from uncertainty quantification to collaborative approaches.

The duration of the learning phase is critical for using learning systems in the field of comfort and infotainment. In this regard, the presented algorithms need to be evaluated with further field studies before being deployed in series production vehicles.

ACKNOWLEDGMENT

The authors would like to thank the participants of the studies and BMW Group for providing the drive data and the equipment, including the test vehicles. All location and drive data was anonymized to ensure the participants' privacy.

REFERENCES

[1] S. Haas, K. Wiesner, and T. C. Stone, "Car ride classification for drive context recognition," in MOBILITY 2014, The Fourth International Conference on Mobile Services, Resources, and Users, 2014, pp. 61–66.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in Handheld and ubiquitous computing. Springer, 1999, pp. 304–307.

[3] M. Bazire and P. Brzillon, "Understanding context before using it," in Modeling and Using Context, ser. Lecture Notes in Computer Science, A. Dey, B. Kokinov, D. Leake, and R. Turner, Eds. Springer Berlin Heidelberg, 2005, vol. 3554, pp. 29–40. [Online]. Available: http://dx.doi.org/10.1007/11508373_3

[4] G. D. Abowd et al., "Towards a better understanding of context and context-awareness," in Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, ser. HUC '99. London, UK, UK: Springer-Verlag, 1999, pp. 304–307.

[5] A. Schmidt, "Ubiquitous Computing - Computing in Context," Ph.D. dissertation, Lancaster University, November 2002.

[6] O. Mazhelis, "Real-time recognition of personal routes using instance-based learning," in IEEE Intelligent Vehicles Symposium (IV 2011), 2011, pp. 619–624.

[7] J. Froehlich and J. Krumm, "Route prediction from trip observations," in Proceedings of the Society of Automotive Engineers (SAE) 2008 World Congress, SAE Technical Paper 2008-01-0201, April 2008, pp. 1–13.

[8] D. Tiesyte and C. S. Jensen, "Similarity-based prediction of travel times for vehicles traveling on known routes," in Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ser. GIS '08. New York, NY, USA: ACM, 2008, pp. 14:1–14:10.

[9] A. Brilingaite and C. S. Jensen, "Online Route Prediction for Automotive Applications," in Proceedings of The 13th World Congress and Exhibition on Intelligent Transport Systems and Services (ITS 2006), London, October 2006, pp. 1–8.

[10] K. Torkkola, K. Zhang, H. Li, H. Zhang, C. Schreiner, and M. Gardner, "Traffic Advisories Based on Route Prediction," in Proceedings of Workshop on Mobile Interaction with the Real World, 2007, pp. 33–36.

[11] O. Mazhelis, I. Žliobaite, and M. Pechenizkiy, "Context-aware personal route recognition," in Proceedings of the 14th international conference on Discovery science, ser. DS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 221–235.

[12] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in Data Engineering, 2002. Proceedings. 18th International Conference on, 2002, pp. 673–684.

[13] K. Laasonen, "Route Prediction from Cellular Data," in Proceedings of the Workshop on Context-Awareness for Proactive Systems (CAPS). Helsinki, Finland: University Press, 2005, pp. 147–158.

[14] K. Tanaka, Y. Kishino, T. Terada, and S. Nishio, "A destination prediction method using driving contexts and trajectory for car navigation systems," in Proceedings of the 2009 ACM Symposium on Applied Computing, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 190–195.

[15] D. Ashbrook and T. Starner, "Using gps to learn significant locations and predict movement across multiple users," Personal Ubiquitous Comput., vol. 7, no. 5, Oct. 2003, pp. 275–286.

[16] J. Krumm and E. Horvitz, "Predestination: Inferring destinations from partial trajectories," in Ubicomp, 2006, pp. 243–260.

[17] C. Zhou, N. Bhatnagar, S. Shekhar, and L. Terveen, "Mining personally important places from gps tracks," in Data Engineering Workshop, 2007 IEEE 23rd International Conference on, 2007, pp. 517–526.

[18] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen, "Discovering Personally Meaningful Places: An Interactive Clustering Approach," ACM Trans. Inf. Syst., vol. 25, no. 3, July 2007.

[19] J. H. Kang, W. Welbourne, B. Stewart, and G. Borriello, "Extracting places from traces of locations," in Proceedings of the 2Nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots, ser. WMASH '04. New York, NY, USA: ACM, 2004, pp. 110–118.

[20] T. Stone, O. Birth, A. Gensler, A. Huber, M. Jänicke, and B. Sick, "Location based learning of user behavior for proactive recommender systems in car comfort functions," in Proceedings of INFORMATIK 2014, GI-Edition - Lecture Notes in Informatics (LNI), 2014, pp. 2121–2132.

[21] J. B. Macqueen, "Some methods of classification and analysis of multivariate observations," in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[22] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in KDD, vol. 96, no. 34. AAAI Press, 1996, pp. 226–231.

[23] M. Lewandowsky and D. Winter, "Distance between sets," in Letters to nature. nature publishing group, 1971, pp. 34–35.

[24] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," ACM Transactions on Mathematics Software, vol. 3, no. 3, September 1977, pp. 209–226.

[25] R. W. Sinnott, "Virtues of the Haversine," Sky and Telescope, vol. 68, no. 2, 1984, pp. 159+.

[26] J. Han, M. Kamber, and A. K. H. Tung, "Spatial clustering methods

in data mining: A survey," in Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS, H. J. Miller and J. Han, Eds. Taylor and Francis, 2001, pp. 201–231.

[27] A. Schmidt, M. Beigl, and H.-W. Gellersen, "There is more to context than location," Computers & Graphics, vol. 23, no. 6, 1999, pp. 893–901.

[28] D. W. Scott and S. R. Sain, "Multi-Dimensional Density Estimation". Amsterdam: Elsevier, 2004, pp. 229–263.

Hands-on Smart Card User Interface Research, Development, and Testing

Markus Ullmann* † and Ralph Breithaupt*
* Federal Office for Information Security
D-53133 Bonn, Germany
Email: {markus.ullmann ralph.breithaupt}@bsi.bund.de
† University of Applied Sciences Bonn-Rhine-Sieg
Institute for Security Research
D-53757 Sankt Augustin, Germany
Email: markus.ullmann@h-brs.de

***Abstract*—The latest advances in the field of smart card technologies allow modern cards to be more than just simple security tokens. Recent developments facilitate the use of interactive components like buttons, displays or even touch-sensors within the card's body thus conquering whole new areas of application. With interactive functionalities the usability aspect becomes the most important one for designing secure and popularly accepted products. Unfortunately, the usability can only be tested fully with completely integrated hence expensive smart card prototypes. This restricts severely application specific research, case studies of new smart card user interfaces and the optimization of design aspects, as well as hardware requirements by making usability and acceptance tests in smart card development very costly and time-consuming. Rapid development and simulation of smart card interfaces and applications can help to avoid this restriction. This paper presents a rapid development process for new smart card interfaces and applications based on common smartphone technology using a tool called SCUID^{Sim}. We will demonstrate the variety of usability aspects that can be analyzed with such a simulator by discussing some selected example projects.**

Keywords–Smart Card; Smart Card User Interface Design, Interactive Smart Card Applications; Rapid Prototyping; Simulation; Testing; Usability.

I. INTRODUCTION

Today, smartphones belong to the most widely used electronic consumer devices for communication, infotainment and entertainment. High end systems are equipped with a powerful processor, graphics processor, camera, high resolution display as well as a lot of sensors and support various communication technologies, thus providing a powerful combination of convenience and versatile possibilities. That is why more and more applications that require a high level of security like banking or electronic shopping are becoming tools for everyday use. Therefore, a secure element on the device is included to store the cryptographic keys and to perform cryptographic operations. Unfortunately, it is very challenging to combine high security elements with complex consumer electronics optimized for convenience with their different open APIs, communication interfaces and update mechanisms, as jailbreaks and other successful attacks on employed security measures have shown in the past.

If security is the dominant issue, e.g., for personal identification, authentication, access control, banking, pay-tv, crypto services, etc., another system progression takes place based on smart cards, which consists mainly of a secure element embedded in a smart card body. Smart cards are superior in the area of low cost, mobility and especially certifiably high security. However, common smart cards are simple security tokens and have no user interface. Every additional authentication process requires external devices like, e.g., keypads for entering passwords, which by themselves are potentially vulnerable against side channel attacks, eavesdropping, etc. It would be ideal to combine the security level of a smart card with the convenience of a smartphone to keep everything "on card". Recently developed interactive components allow the integration of input devices, like buttons, keypads or touch based gesture interfaces as well as output devices like displays and LEDs directly into a smart card.

With such interactive functionalities the usability aspect becomes the most important one for designing a usable smart card and adds many new demands to the development process. Now, aspects like the adequate size of a button, the visibility of a touch interface, the resolution, contrast and speed of a display and the overall design of the card have to be addressed as well as an appropriate hardware/software-codesign to ensure clear user guidance and high overall usability. This can only be achieved by conducting extensive field tests with as many people as possible. Creating the necessary card prototypes with the complete design and full hardware and software functionality can be very expensive and time-consuming, which makes usability centered security research difficult. This is the motivation for SCUID^{Sim} (**S**mart **C**ard **U**ser **I**nterface **D**evelopment **Sim**ulator): to support the development and evaluation of smart cards with user interfaces. In this paper, we present an alternative approach to allow all the necessary testing in order to determine the requirements for design, hardware components and the software without the need to build costly prototypes. By using common smartphones as a development and evaluation platform, almost all user related aspects can be investigated by simulating the "look & feel" of a new smart card design before any real hardware integration is needed.

SCUID^{Sim} is an android application and therefore usable on a wide range of smartphones, which combine all the necessary hardware input/output components as well as communication links, cryptographic services, the processor power and memory needed for simulating a large variety of current and future smart card interfaces and applications in a single compact device. With SCUID^{Sim} the visible aspects of a multi-component smart card can be designed on the smartphone. Based on a simple SCUID^{Sim}-API, user defined card applications can be executed while SCUID^{Sim} simulates the behavioural properties of all interactive components. New requests and requirements

can be implemented, simulated and evaluated instantly. This way SCUID^{Sim} supports detailed requirement engineering for software as well as hardware and the development of new user interface concepts hand in hand. This is especially useful for the design and integration of new usable user centric security algorithms in smart cards. SCUID^{Sim} was firstly published at The Fourth International Conference on Ambient Computing, Applications, Services and Technologies (AMBIENT 2014) [1]. In this enhanced journal version, the focus is on smart card user interface research, development and testing using SCUID^{Sim}.

In recent years much effort has been made in order to integrate segmented displays into smart cards. Most prototypes used electrophoretic display technology, as shown in Figure 3, known from ebook readers. But this direction was no success story to date. High costs were one reason - but the main problem was its typical update delay of 1 second, which for small displays did not get enough public acceptance. From this experience many questions emerged, like: how fast does such a display have to be in order to be acceptable for certain applications? What resolution and contrast is necessary for adequate usability? Such usability centred questions are very hard to investigate with real smart card prototypes.

SCUID^{Sim} is a framework to initially develop and test new usability approaches very quickly prior to the development of the smart card hardware. This way all user acceptance related issues can be investigated and optimized resulting in a detailed requirement list for all hardware components. It was our goal to significantly increase the speed and efficiency of the development and evaluation of new interactive smart card concepts.

The following sections of this paper are organized as follows: Section II starts with a description of related work. Section III provides a brief overview of the software architecture of SCUID^{Sim} and its functionality. Next, Section IV describes applications of SCUID^{Sim} for the design of smart card user interfaces. We focus our attention on different LED matrix based displays which are very easy to handle, cheap and flexible as an example how the possibilities and challenges of such a technology can be investigated with SCUID^{Sim}. We present the evaluation of various interaction concepts like animated symbols, scrolling text and even rapid serial visual presentation displays for long text passages in regard to the constraints of a smart card. As input mechanism, we present a one-character display input device and in addition user inputs based on touch-gestures. Finally, in Section V we summarize our results.

II. Related Work

The first research and development projects investigating the idea to integrate input and output elements in smart cards go back as far as the late 1990s, see [2]. With the advances in low power and low profile embedded technologies many different component technologies have been successfully developed and integrated in ID1-compatible smart cards during the last decade. Primarily, a variety of display types and buttons, even fingerprint scanners, are discussed for integration, see [3] and [4]. Moreover, in [5] smart cards with an integrated display as security enforcing component are introduced. A first approach to integrate a 2D on-card gesture input sensor, implemented as capacitive touch matrix, is introduced first in [6]. It has also been an important topic for public funding in many countries (e.g., the INSITO-project of the German Federal Office for Information Security (BSI) and the SECUDIS-project of the German Ministry of Education and Research, see [7] and [8]). Despite all the effort and the growing number of available components, interactive smart cards have not yet been used in many real applications. Among other reasons this is due to high production costs and the much higher complexity of such smart cards. With the recent advances in printed electronics capacitive sensors have become a widely accepted standard technology and even printed displays are available today, see [9], [10], and [11]. But the complexity issue is still a serious obstacle on the way to the final product. At least regarding the system integration issues of combining several hardware components there have been approaches for rapid prototyping tools. One of the first was the FlexCOS system suggested by Beilke et al. [12], which uses FPGAs for a very flexible and rearrangeable interface to connect separate component prototypes into one complete system. Although this approach became a standard procedure for many manufacturers and researchers, it only covers the technological aspects. Such functional prototypes are much too bulky and fragile to conduct real world tests with many people in real application scenarios outside the lab. The usability aspects that first and foremost define how the smart card should interact, and therefore, what the requirements for the hardware and software components really are can not be tested without fully integrated and designed card prototypes. Unfortunately, each version of real prototypes to test for user acceptance requires huge expenses of time and money. This lack of end-user centered rapid prototyping tools was the starting point for the development of the SCUID^{Sim} tool. Simulation of user interfaces was very popular in the beginning of ubiquitous computing. One approach was the iStuff toolkit to support the development of user interfaces for the post-desktop age for multiple displays, multiple input devices, multiple systems, multiple applications and multiple concurrent users, see [13]. Alternative technologies were developed by the Stanford Interactive Workspaces project for multi-person and multi-device collaborative work settings, see [14]. To the best of our knowlege, SCUID^{Sim} is the first approach to model, simulate and analyze user interfaces for (contactless) smart cards.

III. SCUID^{Sim} Architecture

SCUID^{Sim} consists of two modules: a card designer that enables a flexible but simple arrangement of smart card layouts based on preconfigured components and a card simulator. In the card simulator, such a card layout can be paired with a smart card application in a real time simulation. It was a design decision to separate the card design process and the card simulation process in two independent software modules. Figure 1 illustrates the SCUID^{Sim} architecture.

A. Card Designer

The card designer is a simple tool to engineer smart card layouts. Figure 2 gives an overview of the available components in the current version of SCUID^{Sim}. Currently, the following predefined components are supported: push buttons, segmented displays (7- and 14-segments), matrix displays (RGB, greyscale and black & white), LEDs, n $\times$ m LED-matrixes, 2D-touch sensors, image boxes and the overlay

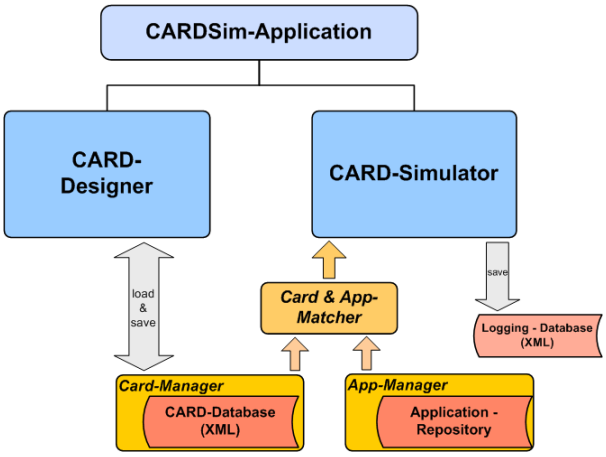


Figure 1. Overview of the SCUID^{Sim} software architecture.

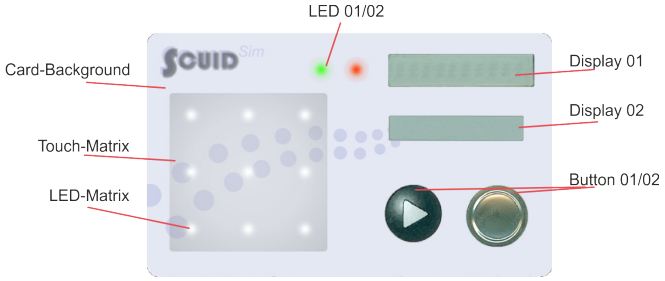


Figure 2. Available card components (in the card designer)

image of the smart card. There are also non-visible components like acceleration sensors that are automatically available to all cards if the used android smartphone is supporting it. With this initial set of predefined components, SCUID^{Sim} can already simulate a huge variety of smart card layouts. Figure 3 depicts a real card prototype opposite to a replicated design of this card within SCUID^{Sim}. This figure illustrates the very realistic replication capabilities of our tool.

Within the card designer, the properties of each component like position & size can easily be controlled via simple finger gestures commonly known from many other mobile applications. Additional properties like the appearance of the component (overlay image), a color modifier (to the overlay image, in RGB and alpha for transparency) or component specific properties like X/Y-resolution of a matrix display, or the update delay time for a display component can be set in a component property page that is dynamically generated

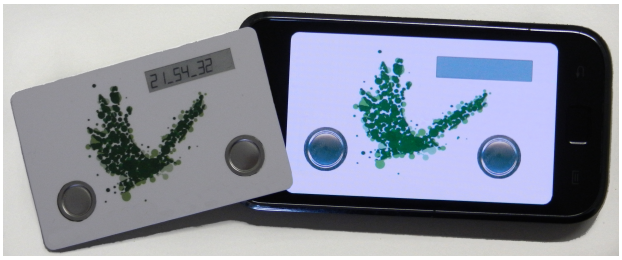


Figure 3. Confrontation real - and simulated card layout within SCUID^{Sim}

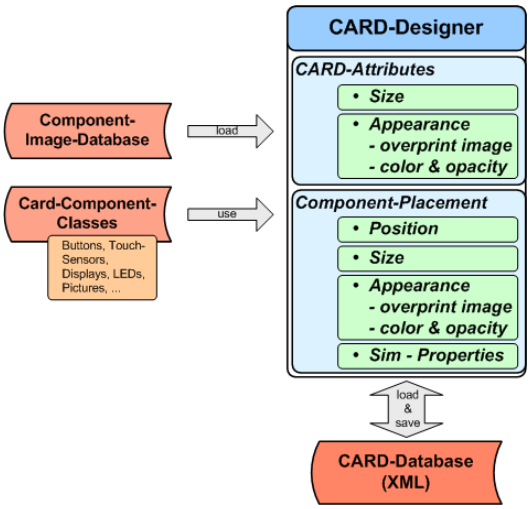


Figure 4. Software architecture of the card designer module

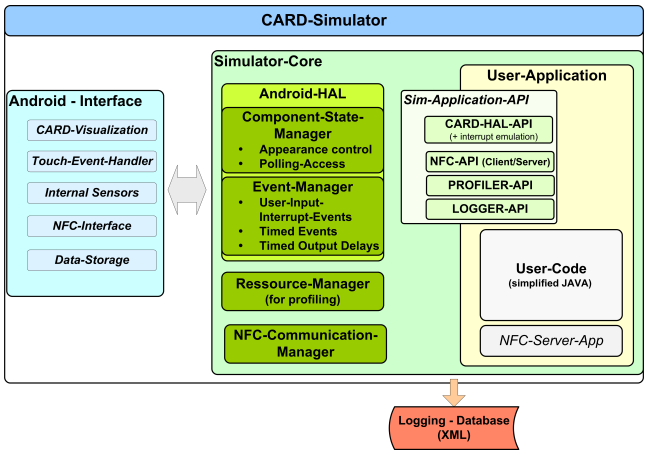


Figure 5. Software architecture of the card simulator module

based on all the properties of a selected card component. Each card component, its properties and its specific simulated behaviour (e.g., delay of the visual update) is defined in the respective class within the component library of SCUID^{Sim}. To add new components or behavioural functionality to this library, the developer simply inherits and modifies the provided component base class. All administrative support like the list of available component types and the components property page are generated “on the fly”. The complete card design can be loaded from and saved to a card library in a XML-format that can be read and edited outside SCUID^{Sim} with all existing standard XML-viewers/editors. New overlay images and even new components are easily added to the designer. Figure 4 depicts the software architecture of the card designer.

B. Card Simulator

The two main objectives of the card simulator are to provide a flexible framework for the development and evaluation of card applications and to simulate an user interaction with a realistic “look & feel”-experience. For creating card applications, the card simulator offers a simple API in order to access the interactive components of the simulated card. In

order to keep the application as close to a real card program as possible the API allows input components to be polled and provides a simulated interrupt event handling. The concept of the API is based on the intention to shield the application developer from Android Java specific constructs in order to facilitate application code that can easily be transferred to real smart cards. In addition, the card simulator consists of a ressource manager module for simple profiling purposes as well as a flexible XML-based logging system. Since most applications for contactless smart cards imply a communication to a reader/server via NFC (ISO 14443) the card simulator offers an interface to the real NFC component of a smartphone. This way the simulated card can also be used in the targeted environment. If the real NFC component cannot be used, the framework allows the execution of NFC server applications in order to also simulate the reader functionality. A manual describing the usage and programming of applications can be found in [15]. Figure 5 depicts the software architecture of the card simulator module.

IV. Using SCUID^{Sim} for Smart Card User Interface Development and Testing

In this section, we present some examples how usability aspects of interactive smart cards can be analyzed and optimized by using SCUID^{Sim}. Typically, contactless cards follow the ISO 14443 specification, see [16]. This means that contactless smart cards usually have no battery. They are powered by the magnetic field of the terminal device. So, the available energy on real contactless smart cards for powering additional components is very limited and energy is only available if the card is in the activation distance of a terminal. The most used smart card format is ID-1 according ISOIEC 7816 [17]. This format is restricted to 85,60 mm x 53,98 mm. This makes it obvious that there is only very restricted space for additional on-card input and output components on such a smart card.

The main idea of SCUID^{Sim} is to rapidly design smart card layouts including on-card input and output devices and related applications according to this restricted dimensions and resources hand-in-hand. These capabilities can be used to design and explore new applications and to perform user studies on standard smartphones before any real smart card prototype is produced. Moreover, these capabilities of SCUID^{Sim} can be used for requirement engineering of real cards.

Our design and simulation framework, in combination with the wide availability of simulation platforms (android smartphones), made it possible for our students to quickly and easily investigate isolated usability aspects. The following brief examples of our latest studies should provide an adequate overview over the kind of analyses that can be performed with our simulation approach. Although we have performed first user studies, we do not give priority to this topic in this paper. But we include some interesting findings of these studies. The group consists of thirty unspecific persons with different levels of knowledge of information technology (in age 20 - 70: 11 female, 19 male). Obviously, there are no adequate user cross sections and no statistical relevant number of attendees.

A. Corporate Design Aspects

An important aspect in the smart card business is the combination of functional elements with the corporate design. Here the SCUID^{Sim} card designer can be easily used to design different layouts of the card body including any kind of branding as well as visual user guidance elements. These designs can be thoroughly evaluated with the simulator to explore the influence of the branding to the usability. Specifically: design of the interactive components (size, look and placement), the necessary user guidance elements and the corporate design elements to find suitable combinations in a way that the handling of the card is always clear to most of the targeted customers. Figure 3 illustrates this issue.

B. On-Card Output-Components

Most outputs on electronic devices are usually performed with optical segment- or matrix-displays. Due to the very restricted form factor of smart cards the displays itself are very restricted in their dimensions. Here physiological studies are very interesting, which were already performed in the 80's and 90's with text display formats. These display formats are rapid serial visual presentation (RSVP), in which each word is displayed sequentially at the same place on the display screen, and scrolled text, in which 13 characters are scrolled continuously from right to left or left to right across the screen. The dynamic and continuous presentation of text in this both displays requires smaller eye movements compared to usual monitors. This change in eye-movement in contrast to classical displays is believed to be responsible for higher reading rates in contrast to reading rates of usual monitors. The detailed results in the studies differ a little bit, see [18], [19], and [20], but these types of text presentation opens the perspective for displaying larger texts in restricted smart card displays.

General properties for displays are the resolution, contrast, color of the characters and background color for smart card displays, too. But besides that, the visibility of the display within the card body is an important issue. See the difference of the optical awareness of the displays between Figure 10 and Figure 12. In Figure 12, the LED matrix is included in a box and has a different background color compared to the card body. That is an important usability issue.

Besides the general display requirements and the awareness of the display, the application of the smart card display is important. Is it intended for displaying one time passwords (strings of up to 16 random characters), control instructions, short text outputs, user feedbacks, telephone numbers, long text of a few sentence, graphical symbols or others? What is the real use case of the display and which information has to be presented to the user in an adequate and readable manner? A further issue is the required speed of the display from an application perspective. So, the real display requirements are application specific and have to be tested in regard of user readability and comprehension.

1) Segmented Text Displays with Low Printing Rate: Figure 3 presents our first project in which we simulated an existing smart card prototype in order to start with ground-truth tests to determine the comparability of real and simulated cards. These cards were equipped with a standard 14-segment display component with 10 characters based on electrophoretic display technology and two buttons. The real display had good properties like high contrast and low power consumption, but unfortunately, a very low refresh rate of 1 word/update per second. It was surprising how similar the overall look and feel of the simulated card actually was. It turned out in our tests that the bigger body of the smartphone is not a big issue

when it comes to usability aspects. The biggest issue of the simulated card was still the very long delay of the display. From this point on we were able to change the key parameters in the simulation: update delay, contrast, size of the display in order to find out what the minimal properties of such a display should be in order to be acceptable for most of our test subjects.

2) LED-Matrix Display for Displaying One Symbol: Complex displays, especially matrix-displays will always be a problem for smart cards. Even if the costs for the display itself can be reduced significantly, there will always be the need for display drivers, which means more complexity and costs. Also, a bigger display size would be good for usability but an increasing challenge in the integration process. Looking for alternatives, the old concept of using LEDs to build low resolution displays came to mind. LEDs are quick (even animations are possible), relatively cheap and easy to control and have the advantage that they can be mixed with other components. In that way, LED-displays can be almost as big as the card itself. On the other hand, the power consumption of LEDs is an issue and depends largely on the number of used LEDs and their brightness. In order to find out, if LED-displays could actually be a practicable alternative in smart cards, we used SCUID^{Sim} to determine the requirements regarding speed, brightness, color, size and resolution and investigated appropriate interface concepts regarding fonts, symbols, animation, user guidance and feedback.

We started with the lowest configuration humans generally can read comfortably: a 3 × 5 LED-matrix.

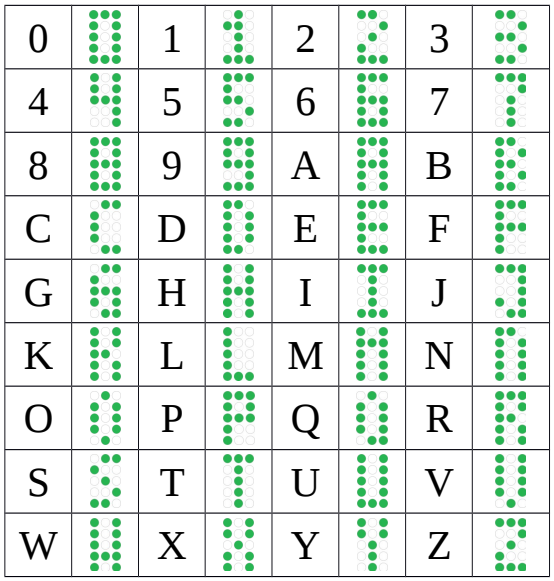


Figure 6. Used 3 × 5 LED matrix font

Figure 6 shows that our chosen font for a 3 × 5 LED matrix works quite well for digits, while on the other hand, some characters are only poorly distinguishable (like: U, W, H, M, O or Q). Lower letters worsen the problem even more. This means if only digits are processed a 3 × 5 LED matrix seems to be sufficient. But, if letters should be processed higher resolutions, like in a 4 × 5, 5 × 5, 4 × 7, or 5 × 7 LED matrix, displays are needed to achieve better character readability. That has to be analyzed.

But, which character representation should be chosen? There are no standard fonts and tests are needed to achieve distinct human readability. Figure 7 shows this difficulty for the number nine in a 5×7 LED matrix setting. But this holds for the whole font and has to be analyzed seriously.

Figure 7. Digit 9 in five different illustrations in a 5×7 font

Next, the principle illustration facilities of a 3 × 5 LED matrix display are presented. Static characters:

1) Characters, e.g., alphabet shown in Figure 6
2) Special characters, e.g., dice symbols shown in Figure 8
3) Symbols, e.g., arrows, rectangle, box, horizontal and vertical lines, etc.

Animated symbols:

4) Special characters, e.g., falling dice symbols
5) Symbols, like a falling arrow (picture frequency 200 ms), curtain up (picture frequency 200 ms), curtain down (picture frequency 200 ms) and rotary dots (dot frequency 200 ms) shown in the first row from left to right in Figure 9 and helix construction (sequentially build up dot by dot with dot frequency 200 ms), helix destruction (sequentially build up dot by dot with dot frequency 200 ms), o.k. symbol (sequentially build up dot by dot with dot frequency 200 ms) and fail symbol (sequentially build up dot by dot with dot frequency 200 ms) shown in the second row from left to right in Figure 9.

Not surprisingly, animated symbols like falling arrows and rectangles, dynamic curtains, circling dots, etc. seem to be very intelligible to the user and compensate for the low resolution to some degree. Animated symbols seem to be a suitable

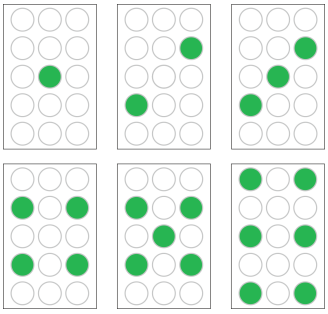


Figure 8. Digits 1 up to 6 as dice symbol

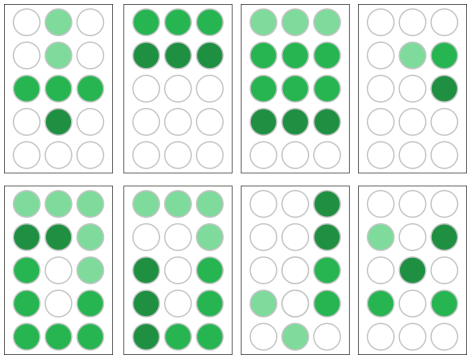


Figure 9. Animated symbols

alternative to text output to indicate card states and to give feedback information to the user.

This project showed that even a very restricted 3×5 LED-matrix display enables the presentation of a large range of characters and symbols, especially, when static and dynamic (animations) effects are exploited. Our first test results indicated that even symbols need to be chosen very carefully and have to be explained to the user in detail in order to achieve a high recognition rate. If it is possible to use symbols which are intelligible to all, they should be applied in any case.

Additionally, we tried to output short words (e.g., on, off, etc. ...) by sequentially displaying the characters of the word (rapid serial visual presentation of characters). The test users had enormous problems to read and identify even very short words depicted as sequence of letters, when they did not know the displayed word beforehand. In consequence, this approach does not seem to be suitable for displaying words. A known alternative for displaying text in a LED-matrix is of course based on scrolling text.

3) LED-Matrix Display for Srolling Text: The lower bound from a human readability perspective seems to be more or less displaying at least 2 clearly separated characters at a time. For a 5×5 font a 11×5 LED matrix display is needed to fulfil this requirement. Such a card layout is shown in Figure 10. This card is intended to perform user tests of scrolling text. Therefore, this card is equipped with a slider component and buttons for card configuration. Especially, the slider component provides user controlled repeatability of already shown text (here by swiping the finger to the left).

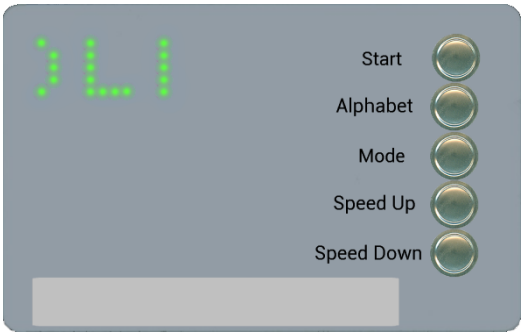


Figure 10. Layout of a smart card with 11×5 LED matrix display and slider component to perform user tests of scrolling text

Concerning the real application of the LED-matrix display numerous questions arise: Which font with which size should be used, what is the adequate speed for the displayed information and how many characters are to be displayed each time? These are specific questions, which we started to investigate in our latest studies.

Due to readability issues the majority of our testing group would prefer higher resolution fonts like 4×7 or 5×7 instead of 3×5, 4×5, or 5×5 and conceive the slider component as very helpful especially if a sequence of digits (e.g., Tel-Nr.: "0228999582") or random characters (e.g., "H7FZ84Q2H07") is shown. Furthermore, the impression is given that short texts (e.g., "PIN") or longer semantic texts (e.g., "BITTE PIN EINGEBEN FUER ANMELDUNG") can be read quite well. It turned out that a scroll speed of 52 characters per minute (cpm) instead of 95 or 38 cpm is comfortable for most of our test subjects.

If longer texts have to be displayed, scrolling text with LEDs has its limits. For such applications, rapid serial visual presentation seems to be much more adequate as we will present in the following example.

a) Matrix-Display for Rapid Serial Visual Presentation: Rapid Serial Visual Presentation (RSVP) uses the phenomenon that presenting a text word after word while keeping the centre of each word on the same spot in the display reduces the necessity for eye-movements and thereby can speed up the reading speed significantly. Studies have shown that with little training reading speeds of 1000 words per minute (wpm) and more are possible. Hence, this technique could be the ideal solution for smart cards with a one word display in applications where it could be necessary to read longer legal or technical instructions.

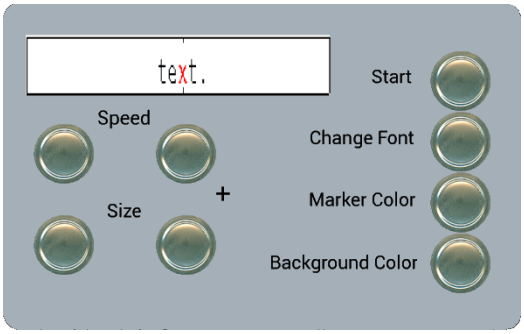


Figure 11. Layout of a smart card with a display for rapid serial visual presentation to perform user tests

Figure 11 shows an example for user tests of rapid serial visual presentation. The simulated smart card is equipped with a fast matrix display and buttons to configure the parameters of the test.

Concerning the real application of a display for RSVP, precise requirements have to be analyzed. This includes the useable font and the size, adequate speed for the displayed information, color of the central character of a word to fix the eyes to this position etc.

Result of the case study with the test group: The majority of the group enjoyed the following text configuration: largest font of 20 point instead of 16 or 12 point and a printing

rate of about 150 words per minute (wps) instead of 140 wps or 270 wps, font latin instead of courier and red color for the character marker, see Figure 11. Long text (e.g., "DAS IST EIN TEST TEXT. WIR KOENNEN KURZE WOERTER UND LANGE WOERTER WIE DONAUDAMPFSCHIFFFAHRTSKAPITAEN NUTZEN. WIR KOENNTEN UNS AUCH EINEN BESSEREN TEXT AUSSUCHEN") is very well conceivable. But a problem arises if words do not fit completely into the display (e.g., "DONAUDAMPFSCHIFFFAHRTSKAPITAEN"). Then the word has to be split into parts and the parts have to be displayed sequentially. Humans can read complete words in a RSVP very well. But this is not the case if only parts of words are sequentially displayed very rapidly.

b) Matrix-Display for Graphics: The most flexible (and technically most challenging) display type for a smart card is of course a high resolution matrix display. ID-Cards or driver licence cards are typically equipped with a printed photo of the face of the legitimate user. This static photo binds the card permanently to one user and can only illustrate one perspective. A matrix display would make such a smart card reusable while also providing different views of the face and additional information about the user. This is very helpful for a manual inspection of the legitimate user. But again questions arise: which display resolution is necessary and what is the required update rate? Another application of a matrix display is to show bar codes. Bar codes can be used for optical data transmission from a smart card to a terminal for automatic data recognition and data processing. Which display size and which resolution is needed? Again, this can be easy implemented and tested with SCUID^{Sim}.

C. On-Card User Input Components

A complete on card interface needs input components as well as displays. Since in early studies almost all known button technologies with a sensible pressure point or any other kind of haptic feedback share the disadvantage to reduce the physical integrity of the smart card body, capacitive sensors are an almost ideal alternative. They are cheap and easy to integrate into a smart card and can even be used as 2D-touch sensors for complex gesture inputs - they just do not provide any kind of feedback. With such input components the aspect of visual design and user guidance is the most important issue. Also, parameters like the reading rate and resolution can be a crucial factor for the success of a specific application. Is the input component intended for the use of numbers like PINs or one time passwords (strings up to 16 random characters) or control instructions for applications? So, the real input requirements are again application specific and have to be tested in regard of user understanding and awareness.

1) Buttons: The test card in Figure 10 uses the preconfigured buttons of SCUID^{Sim}. But what is an adequate button design regarding size, distance between buttons for precise operation, color, or the user feedback in case the button is pressed? Again these design issues have to be analyzed in future studies.

2) One Character Display Input: In Section IV-B2, we have shown that a 3×5 LED matrix can be sufficient if only digits are used. Figure 12 shows a smart card with a 3×5 LED matrix for illustrating only one-character and a slider component for user input. With wiping gestures the user can scroll through the characters of the alphabet and a long touch (e.g., $\geq$ 1 second) selects the currently displayed character for input. User control and feedback (e.g., about the current position within a PIN, error or success messages, etc.) can be given with a variety of generally accepted static or even animated symbols. Such control and feedback concepts can again be implemented and tested based on SCUID^{Sim} with users under conditions that are in many aspects very similar to real cards. In [1], a case study on user authentication based on a PIN is given for the card configuration shown in Figure 12.

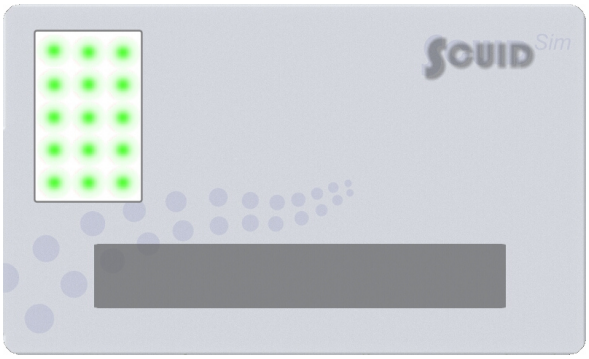


Figure 12. Layout of a smart card with a 3×5 LED matrix display and slider component to perform user inputs

3) One Character Gesture Input: In recent years, 2D finger-gestures became the favoured control concept for operating (smart-)phones. This very flexible, intuitive and therefore widely accepted input interface has also the huge advantage to be integrable into a smart card in a relatively cheap and easy way. In [6], a first real smart card prototype with such a gesture input component is presented. The sensor is a capacitive touch matrix with the size of 40 x 40 mm and is able to calculate the position of a finger-sensor contact with a resolution of 6-7 bits (which results in 64 to 128 distinguishable positions for each axis, or about 80 DPI) with a sampling rate of one point every 16ms. Gestures are recognized by the touch sensor as a time series of touch point coordinates within the active area. In that way, stroke directions and complex gestures can be detected. In contrast to a keypad a gesture interface is not restricted to input digits or characters. But it requires some form of on-line character recognition. On-line character recognition has to process and recognize the handwriting in real-time, ideally while the writing is still ongoing [21], in order to reduce delays. The process of character recognition can be divided into three general steps:

1) pre-processing of the input character information
2) extraction of the character features and
3) classification of the input character

Due to the primary application of smart cards, the recognition of digits (for the input of a PIN or a OneTimePassword) and control commands to operate a card are very important. But again it has to be analyzed which specific pre-processing, feature extraction and classification mechanisms are adequate for the mentioned characters and can be implemented on a very resource restricted device like a smart card. This task becomes particularly difficult if the card should detect the input of as many people as possible. In the master thesis [22],

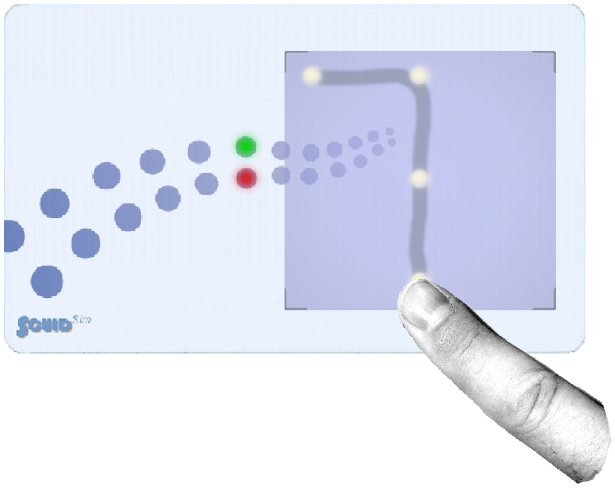


Figure 13. Layout of a smart card with a gesture component to perform user inputs

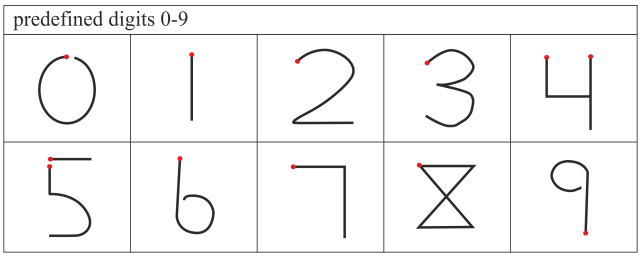


Figure 14. Predefined digits 0 to 9 for easier PIN recognition. A red dot marks the beginning of an episode

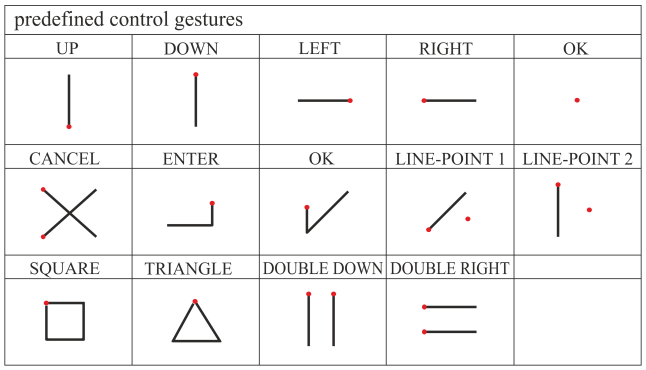


Figure 15. Predefined control gestures. A red dot marks the beginning of an episode

our simulation approach was used to develop and compare suitable gesture recognition algorithms with the focus on the reduced computing capability and memory resources of a smart card with good first results for small and middle sized sets of character and command gestures. Figure 14 shows an example for a set of predefined digits, while Figure 15 shows a subset of predefined control commands that have been used in this study. The concept of predefining the allowed alphabet for simplifying handwriting recognition was firstly introduced by Palm Inc. in their handwriting recognition software called Graffiti. The set of predefined digits [0 - 9] introduced here is similar to a simplified version of Graffiti.

Pre-Processing: The main purpose of pre-processing is to improve the input data to make the recognition process easier and more reliable, e.g., by removing irrelevant information from the sensor input which might have a negative effect on the character recognition [23]. The most important pre-processing techniques for online handwritten characters are: ”filtering”(e.g., ”noise reduction”), ”rotation”, ”size normalization” and ”filling”. Noise produced by the touch sensor may result in duplicate or erratic data points [24]. Filtering is applied to remove this kind of input data [25]. Rotation is needed to obtain an aligned representation of the inserted gesture. Users do not put in handwritten characters in exact the same size. Especially, when pixel based feature extraction is applied, this can have a negative affect on the character classification. So, size normalization is performed to obtain characters of uniform size [26]. Touch sensors have only a limited sampling rate. Depending on the user’s writing speed touch points may differ in the distance. The pre-processing technique ”filling” eliminates small gaps and holes. There exists a lot of algorithms to deal with filling, e.g., Bresenham’s line algorithm [27]. Further complex pre-processing techniques needed for off-line character recognition such as character isolation, line and word detection, etc., very often have high demands for memory and/or a capable CPU and, therefore, are not suitable in our context.

Feature Extraction: Feature extraction is the process of identifying essential characteristics in the representation of the given characters. Two classes of feature extraction methods are distinguished: structural characteristics and statistical characteristics. Usually, feature extraction methods based on structural analysis provide a high tolerance to distortions and style variations [28]. A statistical analysis extracts statistical distribution of points, e.g., in multiple zones by dividing the character image into several overlapping or non-overlapping sub-images. Next, e.g., the percentage or density of black points in each sub-image [29] is calculated. As an alternative, the distance of black points from a given boundary, such as the upper and lower portion of the character, can be used as statistical feature [30].

Classification: During classification an unknown input character image is assigned to its corresponding character class based on a metric [26]. Due to the needed memory and CPU requirements only following character recognition classes are analyzed.

- Pixel Matching: In general, pixel matching determines to which degree the pixel representation of the given input character image corresponds to a pixel representation of a character of the defined character set [31]. For large character sets pixel matching will not provide high recognition rates because the character classes overlap [32].
- Decision Tree: Decision trees are tree-like graphs constructed of multiple decisions and their possible outcome [33]
- Random Forest: Basically, random forest are a combination of several random trees. The idea goes back to Leo Breiman [34]
- k-Nearest Neighbors Algorithm (k-NN): The k-nearest neigbors algorithms calculates the distance of the

feature vector of the input character image to all sample feature vectors in a character set [35].

Due to the necessary training effort and memory requirements artificial neural networks and support vector machines (SVM) often used for such classification purposes are generally problematic and have not been investigated in this study. To correctly compare recognition results and to identify the most suitable recognition system a fixed test procedure is used. For the class of predefined digits $0 \ldots 9$ the detection rate differs from 65.4% (pixel matching) to 98.1 % (random forest). For control gestures the detection rate differs from 60.8% (pixel matching) to 90% (random forest). Moreover, estimations are given concerning memory consumption and calculation time for the mentioned feature extraction and character recognition algorithms.

Now, SCUID^{Sim} can be used to analyze and optimize feature extraction and character recognition algorithms as well as to determine the needed resources for a given target platform (CPU speed, RAM, etc.), if specific character recognition rates have to be assured. Moreover, SCUID^{Sim} can again be utilized to test the usability aspects and the general acceptance of gesture based on-card authentication concepts with a number of applicants in order to test for aspects like: acceptable size of a touch-interface on a smart card, necessary resolution and speed of a 2D-touch sensor and also visual design aspects for clear user guidance.

D. Technical Implementation of Real Smart Card Prototypes

Obviously, not all analyzed user requirements for input and output components are directly realizable in regard to the current state of technology and the energy and cost constraints. In these cases, some further investigation is necessary in order to find a feasible trade-off. We intentionally analyzed different LED matrix displays instead of modern OLED-displays or printed electronics. Contrary to bendable OLED displays and printed displays, real LEDs are available at the market and our prototypes are technically implementable today.

V. Conclusion

In this paper, we present how the tool SCUID^{Sim} can be used for rapid development and simulation of smart card user interfaces and applications. It is utilizable for early considerations of user handling requirements and overall user acceptance of user interfaces before a time-consuming and costly prototype development has to be started. Especially, card designs and application modifications are performed very quickly in software without any hardware modification. This reduces the need for development of smart card prototypes for early considerations and speeds up the whole development process.

Within this paper different specific input and output approaches are presented. First, different LED $n \times m$ matrix displays are described. Surprisingly, even a very restricted 3×5 LED matrix display enables the presentation of digits and a large range of symbols especially when static and dynamic (animations) effects are exploited. Next, a 5×11 LED matrix is presented for scrolling text. This restricted display enables reading words and short sentences although only two characters were presented at each time in this display format. Surprisingly, this form of text presentation was acceptable for the majority of test users. This is a very interesting result and we will further investigate this type of presentation. For long text passages, a display for rapid serial visual presentation is illustrated. RSVP means that words of a text are displayed sequentially at the same place at a display. Users can read even long text in this kind of display. These results show that even long texts can be displayed in very restricted displays if necessary. Regarding user inputs, we introduce a one character display input based on a 3×5 LED-matrix display and one character input based on flexible gestures. The latter enables the input of control commands as shown in Section IV-C3, too. This concept opens totally new usability concepts of smart cards if acceptable character recognition rates can be achieved. Especially, it is shown that even the analysis of feature extraction and character recognition algorithms can be supported by the tool SCUID^{Sim}.

VI. Acknowledgement

The authors would like to thank our students Anton Buzik, Tim Ludemann, Alexander Kreth and David Sosnitza for their support implementing SCUID^{Sim}, Martin Klöckner for the analysis of feature extraction and character recognition algorithms as well as our colleagues Sven Freud and Christian Wieschebrink for valuable remarks. Also thanks to the anonymous reviewers for the valuable comments.

References

[1] M. Ullmann and R. Breithaupt, "Scuid^{Sim}: A platform for smart card user interface research, development and testing," in The Fourth International Conference on Ambient Computing, Applications, Services and Technologies. AMBIENT 2014. IARIA, www.thinkmind.org, August 24 - 28, 2014 - Rome, Italy, pp. 82–87.

[2] Gerald V. Piasenka, and Thomas M. Fox, and Kenneth H. Schmidt, "Solar cell powered smart card with integrated display and interface keypad," 1998, US patent US5777903.

[3] J. Fischer, F. Fritze, M. Tietke and M. Paeschke, "Prospects and Callenges for ID Documents with Integrated Display," in Proceedings of Printed Electronics Europe Conference, 2009.

[4] Bundesdruckerei, "RFID Security Card with a One-Time Password and LED Display," 2013, www.rfidjournal.com/articles/10512.

[5] M. Ullmann, "Flexible visual display unit as security enforcing component for contactless smart card systems," in Firth International EURASIP Workshop on RFID Technology (RFID 2007), 2007, pp. 87–90.

[6] M. Ullmann, R. Breithaupt, and F. Gehring, "On-card user authentication for contactless smart cards based on gesture recognition," in Proceedings GI Sicherheit 2012, ser. Lecture Notes In Informatiks, no. 108, 2012, pp. 223–234.

[7] BDR, "Secudis Project," 2012, http://www.bundesdruckerei.de/en/684-innovative-high-security-solutions.

[8] "Thin chips for document security," in Ultra-thin Chip Technology and Applications, J. Burghartz, Ed., 2011.

[9] Taybet Bilkay, and Kerstin Schulze, and Tatjana Egorov-Brening, and Andreas Bohn, and Silvia Janietz, "Copolythiophenes with Hydrophilic and Hydrophobic Side Chains: Synthesis, Characterization, and Performance in Organic Field Effect Transistors," Macromolecular Chemistry and Physics, vol. 213, September, 26 2012, pp. 1970–1978.

[10] P. Andersson, R. Forchheimer, P. Tehrani, and M. Berggren, "Printable all-organic electrochromic active-matrix displays," Advanced Functional Materials, vol. 17, no. 16, 2007, pp. pp. 3074–3082. [Online]. Available: http://dx.doi.org/10.1002/adfm.200601241

[11] PaperDisplay, "Printed Display Products," 2013, http://www.paperdisplay.se.

[12] K. Beilke and V. Roth, "Flexcos: An open smartcard platform for research and education," in Proceedings of the 6th International Conference on Network and System Security, NSS 2012, ser. Lecture Notes In Computer Science, no. 7645, 2012, pp. 277–290.

[13] R. Ballagas, M. Ringel, M. Stone, and J. Borchers, "istuff: a physical user interface toolkit for ubiquitous computing environments," in Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2003, pp. 537–544.

[14] J. Borchers, M. Ringel, J. Tyler, and A. Fox, "Stanford interactive workspaces: a framework for physical and graphical user interface prototyping," Wireless Communications, IEEE, vol. 9, no. 6, 2002, pp. 64–69.

[15] BSI, "$SCUID^{Sim}$ manual, version 0.3," 2013.

[16] ISO/IEC, "ISO/IEC 144443 contactless Integrated Circuits Cards, Part 1-4: Physical Characteristics (1), Radio Frequency Power and Signal Interface (2), Initialization and Anticollision (3) and, Transmission Protocol (4)," 2000.

[17] ISO, "ISO-IEC 7816-X Identification Cards - Integrated Circuit Cards with Contacts," 2011.

[18] M. C. Potter, "Rapid serial visual presentation (rsvp): A method for studying language processing," New methods in reading comprehension research, vol. 118, 1984, pp. 91–118.

[19] H.-C. Chen, "Effects of reading span and textual coherence on rapid-sequential reading," Memory & cognition, vol. 14, no. 3, 1986, pp. 202–208.

[20] G. S. Rubin and K. Turano, "Reading without saccadic eye movements," Vision research, vol. 32, no. 5, 1992, pp. 895–902.

[21] A. Drissman et al., "Handwriting recognition systems: An overview," 1997.

[22] M. Klöckner, "Gesture Based On-Card User Authentication for Contactless Smartcards," Master Thesis, University Bochum, Bundesamt für Sicherheit in der Informationstechnik (BSI), 2014.

[23] B. Q. Huang, Y. Zhang, and M.-T. Kechadi, "Preprocessing techniques for online handwriting recognition," in Intelligent Text Categorization and Clustering. Springer, 2009, pp. 25–45.

[24] Charles C. Tappert, Ching Y. Suen and Toru Wakahara, "The State of the Art in On-Line Handwriting Recognition," IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 12, no. 8, August 1990.

[25] N. Arica and F. T. Yarman-Vural, "An overview of character recognition focused on off-line handwriting," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 31, no. 2, 2001, pp. 216–233.

[26] L. Eikvil, "Optical character recognition," citeseer. ist. psu. edu/142042. html, 1993.

[27] J. E. Bresenham, "Algorithm for computer control of a digital plotter," IBM Systems journal, vol. 4, no. 1, 1965, pp. 25–30.

[28] C. Y. Suen, M. Berthod, and S. Mori, "Automatic recognition of handprinted charactersthe state of the art," Proceedings of the IEEE, vol. 68, no. 4, 1980, pp. 469–487.

[29] D. Hunt, "A feature extraction method for the recognition of handprinted characters," Machine Perception of Patterns and Pictures, London, England: The Institute of Physics, 1972, pp. 28–33.

[30] R. M. Brown, "On-line computer recognition of handprinted characters," Electronic Computers, IEEE Transactions on, no. 6, 1964, pp. 750–752.

[31] J. LaViola, "A survey of hand posture and gesture recognition techniques and technology," Brown University, Providence, RI, 1999.

[32] R. Watson, "A survey of gesture recognition techniques," Trinity College Dublin, Department of Computer Science, Tech. Rep., 1993.

[33] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, Classification and regression trees. CRC press, 1984.

[34] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, 2001, pp. 5–32.

[35] R. D. Short and K. Fukunaga, "The optimal distance measure for nearest neighbor classification," Information Theory, IEEE Transactions on, vol. 27, no. 5, 1981, pp. 622–627.

Real-Time and Distributed Applications for Dictionary-Based Data Compression

Sergio De Agostino
Computer Science Department
Sapienza University
Rome, Italy
Email: deagostino@di.uniroma1.it

***Abstract*—The greedy approach to dictionary-based static text compression can be executed by a finite state machine. When it is applied in parallel to different blocks of data independently, there is no lack of robustness even on standard large scale distributed systems with input files of arbitrary size. Beyond standard large scale, a negative effect on the compression effectiveness is caused by the very small size of the data blocks. A robust approach for extreme distributed systems is presented in this paper, where this problem is fixed by overlapping adjacent blocks and preprocessing the neighborhoods of the boundaries. Moreover, we introduce the notion of pseudo-prefix dictionary, which allows optimal compression by means of a real-time semi-greedy procedure and a slight improvement on the compression ratio obtained by the distributed implementations.**

Keywords-data compression; decoding; real time application; distributed system; scalability; robustness

I. Introduction

Real time algorithms are very important in the field of data compression, expecially during the decoding phase, and further speed-up can be obtained by means of distributed implementations. We studied these topics in the context of lossless compression applied to one-dimensional data and preliminary results were presented in [1] and [2].

Static data compression implies the knowledge of the input type. With text, dictionary-based techniques are particularly efficient and employ string factorization. The dictionary comprises typical factors plus the alphabet characters in order to guarantee feasible factorizations for every string. Factors in the input string are substituted by pointers to dictionary copies and such pointers could be either variable or fixed length codewords. The optimal factorization is the one providing the best compression, that is, the one minimizing the sum of the codeword lengths. Efficient sequential algorithms for computing optimal solutions were provided by means of dynamic programming techniques [3] or by reducing the problem to the one of finding a shortest path in a directed acyclic graph [4]. From the point of view of sequential computing, such algorithms have the limitation of using an off-line approach. However, decompression is still on-line and a very fast and simple real time decoder outputs the original string with no loss of information. Therefore, optimal solutions are practically acceptable for read-only memory files where compression is executed only once. Differently, simpler versions of dictionary-based static techniques were proposed, which achieve nearly optimal compression in practice (that is, less than ten percent loss). An important simplification is to use a fixed length code for the pointers, so that the optimal decodable compression for this coding scheme is obtained by minimizing the number of factors. Such variable to fixed length approach is robust since the dictionary factors are typical patterns of the input specifically considered. The problem of minimizing the number of factors gains a relevant computational advantage by assuming that the dictionary is *prefix-closed* (*suffix-closed*), that is, all the prefixes (suffixes) of a dictionary element are dictionary elements [5], [6], [7]. The left to right greedy approach is optimal only with suffix-closed dictionaries. An optimal factorization with prefix-closed dictionaries can be computed on-line by using a semi-greedy procedure [6], [7]. On the other hand, prefix-closed dictionaries are easier to build by standard adaptive heuristics [8], [9]. These heuristics are based on an "incremental" string factorization procedure [10], [11]. The most popular for prefix-closed dictionaries is the one presented in [12]. However, the prefix and suffix properties force the dictionary to include many useless elements, which increase the pointer size and slightly reduce the compression effectiveness. A more natural dictionary with no prefix and no suffix property is the one built by the heuristic in [13] or by means of separator characters as, for example, space, new line and punctuation characters with natural language.

Theoretical work was done, mostly in the nineties, to design efficient parallel algorithms on a random access parallel machine (PRAM) for dictionary-based static text compression [14], [15], [16], [17], [18], [19], [20], [21], [22]. Although the PRAM model is out of fashion today, shared memory parallel machines offer a good computational model for a first approach to parallelization. When we address the practical goal of designing distributed algorithms we have to consider two types of complexity, the inter-processor communication and the input-output mechanism. While the input/output issue is inherent to any parallel algorithm and has standard solutions, the communication cost of the computational phase after the distribution of

the data among the processors and before the output of the final result is obviously algorithm-dependent. So, we need to limit the interprocessor communication and involve more local computation to design a practical algorithm. The simplest model for this phase is, of course, a simple array of processors with no interconnections and, therefore, no communication cost. Parallel decompression is, obviously, possible on this model [17]. With parallel compression, the main issue is the one concerning scalability and robustness. Traditionally, the scale of a system is considered large when the number of nodes has the order of magnitude of a thousand. Modern distributed systems may nowadays consist of hundreds of thousands of nodes, pushing scalability well beyond traditional scenarios (extreme distributed systems).

In [23], an approximation scheme of optimal compression with static prefix-closed dictionaries was presented for massively parallel architectures, using no interprocessor communication during the computational phase since it is applied in parallel to different blocks of data independently. The scheme is algorithmically related to the semi-greedy approach previously mentioned and implementable on extreme distributed systems because adjacent blocks overlap and the neighborhoods of the boundaries are preprocessed. However, with standard large scale the overlapping of the blocks and the preprocessing of the boundaries are not necessary to achieve nearly optimal compression in practice. Furthermore, the greedy approach to dictionary-based static text compression is nearly optimal on realistic data for any kind of dictionary even if the theoretical worst-case analysis shows that the multiplicative approximation factor with respect to optimal compression achieves the maximum length of a dictionary element [9]. If the dictionary is well-constructed by relaxing the prefix property, the loss of greedy compression can go down to one percent with respent to the optimal one. In this paper, we relax the prefix property of the dictionary and present two implementations of the greedy approach to static text compression with an arbitrary dictionary on a large scale and an extreme distributed system, respectively. Moreover, we present a finite-state machine implementation of greedy static dictionary-based compression with an arbitrary dictionary that can be relevant to achieve high speed with standard scale distributed systems. We wish to point out that scalability cannot be guaranteed with adaptive dictionary approaches to data compression, as the sliding window method [24] or the dynamic one [11]. Indeed, the size of the data blocks over the distributed memory of a parallel system must be at least a few hundreds kylobytes in both cases, that is, robustness is guaranteed with scalability only with very large files [14], [26]. This is still true with improved variants employing either fixed-lenght codewords [27], [28] or variable-length ones [29], [30], [31], [32], [33].

Finally, we introduce pseudo-prefix and pseudo-suffix dictionaries and show that the algorithms computing optimal factorizations with suffix-closed and prefix-closed dictionaries still work. The advantage of using pseudo-prefix and pseudo-suffix dictionaries is that we add to an arbitrary dictionary only those prefixes or suffixes needed to guarantee the correctness of the optimal solution. This implies a slight improvement on the compression ratio obtained by the distributed implementations. Moreover, we show the impossibility of real-time optimal factorizations if the dictionary is arbitrary.

In Section II, we describe the different approaches to dictionary-based static text compression. The previous work on parallel approximations of optimal compression with prefix-closed dictionaries is given in Section III. Section IV shows the finite-state machine and the two implementations of the greedy approach for arbitrary dictionaries. Experiments are discussed in Section V. Section VI presents the notion of pseudo-prefix and pseudo-suffix dictionaries, where theoretical and further experimental results are discussed. Conclusions and future work are given in Section VII.

II. Dictionary-Based Static Text Compression

As mentioned in the introduction, the dictionary comprises typical factors (including the alphabet characters) associated with fixed or variable length codewords. The optimal factorization is the one minimizing the sum of the codeword lengths and sequential algorithms for computing optimal solutions were provided by means of dynamic programming techniques [3] or by reducing the problem to the one of finding a shortest path in a directed acyclic graph [4]. When the codewords are fixed-length, with suffix-closed dictionaries we obtain optimality by means of a simple left to right greedy approach, that is, advancing with the on-line reading of the input string by selecting the longest matching factor with a dictionary element. Such a procedure can be computed in real time by storing the dictionary in a trie data structure. If the dictionary is prefix-closed, there is an optimal semi-greedy factorization which is computed by the procedure of Figure 1 [6], [7]. At each step, we select a factor such that the longest match in the next position with a dictionary element ends to the rightest. Since the dictionary is prefix-closed, the factorization is optimal. The algorithm can even be implemented in real time with a modified trie data structure [7].

j:=0; i:=0
repeat forever
  for $k = j + 1$ **to** $i + 1$ **compute**
    $h(k)$: $x_k ... x_{h(k)}$ is the longest match in the k^{th} position
  let k' be such that $h(k')$ is maximum
  $x_j ... x_{k'-1}$ is a factor of the parsing; $j := k'$; $i := h(k')$

Figure 1. The semi-greedy factorization procedure.

The semi-greedy factorization can be generalized to any dictionary by considering only those positions, among the

ones covered by the current factor, next to a prefix that is a dictionary element [6]. The generalized semi-greedy factorization procedure is not optimal while the greedy one is not optimal even when the dictionary is prefix-closed. The maximum length of a dictionary element is an obvious upper bound to the multiplicative approximation factor of any string factorization procedure with respect to the optimal solution. We show that this upper bound is tight for the greedy and semi-greedy procedures when the dictionary is arbitrary and that such tightness is kept by the greedy procedure even for prefix-closed dictionaries. Let $baba^n$ be the input string and let $\{a, b, bab, ba^n\}$ be the dictionary. Then, the optimal factorization is b, a, ba^n while $bab, a, a, ..., a, ...a$ is the factorization obtained whether the greedy or the semi-greedy procedure is applied. On the other hand, with the prefix-closed dictionary $\{a, b, ba, bab, ba^k : 2 \leq k \leq n\}$, the optimal factorization ba, ba^n is computed by the semi-greedy approach while the greedy factorization remains the same. These examples, obviously, prove our statement on the tightness of the upper bound.

III. Previous Work

Given an arbitrary dictionary, for every integer k greater than 1 there is an $O(km)$ time, $O(n/km)$ processors distributed algorithm factorizing an input string S with a cost which approximates the cost of the optimal factorization within the multiplicative factor $(k+m-1)/k$, where n and m are the lengths of the input string and the longest factor respectively [14]. However, with prefix-closed dictionaries a better approximation scheme was presented in [23], producing a factorization of S with a cost approximating the cost of the optimal factorization within the multiplicative factor $(k+1)/k$ in $O(km)$ time with $O(n/km)$ processors. This second approach was designed for massively parallel architecture and is suitable for extreme distributed systems, when the scale is beyond standard large values. On the other hand, the first approach applies to standard small, medium and large scale systems. Both approaches provide approximation schemes for the corresponding factorization problems since the multiplicative approximation factors converge to 1 when km converge to n. Indeed, in both cases compression is applied in parallel to different blocks of data independently. Beyond standard large scale, adjacent blocks overlap and the neighborhoods of the boundaries are preprocessed.

To decode the compressed files on a distributed system, it is enough to use a special mark occurring in the sequence of pointers each time the coding of a block ends. The input phase distributes the subsequences of pointers coding each block among the processors. Since a copy of the dictionary is stored in every processor, the decoding of the blocks is straightforward. In the following two subsections, we describe the two approaches. Then, how to speed up the preprocessing phase of the second approach is described in the last subsection. Next section will argue that we can relax the requirement of computing a theoretical approximation of optimal compression since, in practice, the greedy approach is nearly optimal on data blocks sufficiently long. On the other hand, when the blocks are too short because the scale of the distributed system is beyond standard values, the overlapping of the adjacent blocks and the preprocessing of the neighborhoods of the boundaries are necessary to garantee the robustness of the greedy approach.

A. Standard Scale Distributed Systems

We simply apply in parallel the optimal compression to blocks of length km. Every processor stores a copy of the dictionary. For an arbitrary dictionary, we execute the dynamic programming procedure computing the optimal factorization of a string in linear time [3] (the procedure in [4] is pseudo-linear for fixed-length coding and, even, super-linear for variable length). Obviously, this works for prefix- and suffix-closed dictionaries as well and, in any case, we know the semi-greedy and greedy approaches are implementable in linear time. It follows that the algorithm requires $O(km)$ time with n/km processors and the multiplicative approximation factor is $(k+m-1)/k$ with respect to any factorization. Indeed, when the boundary cuts a factor the suffix starting the block and its substrings might not be in the dictionary. Therefore, the multiplicative approximation factor follows from the fact that $m-1$ is the maximum length for a proper suffix as shown in Figure 2 (sequence of plus signs in parentheses). If the dictionary is suffix-closed, the multiplicative approximation factor is $(k+1)/k$ since each suffix of a factor is a factor.

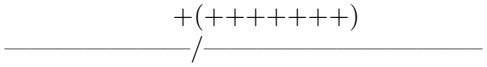


Figure 2. The making of the surplus factors.

The approximation scheme is suitable only for standard scale systems unless the file size is very large. In effect, the block size must be in the order of kilobytes to guarantee robustness. Beyond standard large scale, overlapping of adjacent blocks and a preprocessing of the boundaries are required as we will see in the next subsection.

B. Beyond Standard Large Scale

With prefix-closed dictionaries a better approximation scheme was presented in [23]. During the input phase blocks of length $m(k+2)$, except for the first one and the last one that are $m(k+1)$ long, are broadcasted to the processors. Each block overlaps on m characters with the adjacent block to the left and to the right, respectively (obviously, the first one overlaps only to the right and the last one only to the left). We call a *boundary match* a factor

covering positions in the first and second half of the $2m$ characters shared by two adjacent blocks. The processors execute the following algorithm to compress each block:

- for each block, every corresponding processor but the one associated with the last block computes the boundary match between its block and the next one ending furthest to the right, if any;
- each processor computes the optimal factorization from the beginning of its block to the beginning of the boundary match on the right boundary of its block (or the end of its block if there is no boundary match).

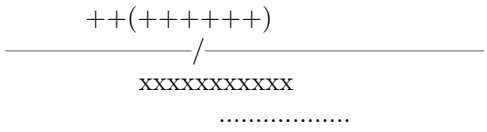


Figure 3. The making of a surplus factor.

Stopping the factorization of each block at the beginning of the right boundary match might cause the making of a surplus factor, which determines the multiplicative approximation factor $(k + 1)/k$ with respect to any other factorization. Indeed, as it is shown in Figure 3, the factor in front of the right boundary match (sequence of x's) might be extended to be a boundary match itself (sequence of plus signs) and to cover the first position of the factor after the boundary (dotted line). Then, the approximation scheme produces a factorization of S with a cost approximating the cost of the optimal factorization within the multiplicative factor $(k + 1)/k$ in $O(km)$ time with $O(n/km)$ processors (we will see in the next subsection how the preprocessing can be executed in $O(m)$ time).

In [23], it is shown experimentally that for $k = 10$ the compression ratio achieved by such factorizarion is about the same as the sequential one and, consequently, the approach is suitable for extreme distributed systems, as we will explain in the next section.

C. Speeding up the Preprocessing

The parallel running time of the preprocessing phase computing the boundary matches is $O(m^2)$ by brute force. To lower the complexity to $O(m)$, an augmented trie data structure is needed [1]. For each node v of the trie, let f be the dictionary element corresponding to v and a an alphabet character not represented by an edge outgoing from v. Then, we add an edge from v to w with label a, where w represents the longest proper suffix of fa in the dictionary. Each processor has a copy of this augmented trie data structure and first preprocess the $2m$ characters overlapped by the adjacent block on the left boundary and, secondly, the ones on the right boundary. In each of these two sub-phases, the processors advance with the reading of the $2m$ characters from left to right, starting from the first one while visiting the trie starting from the root and using the corresponding edges. A temporary variable t_2 stores the position of the current character during the preprocessing while another temporary variable t_1 is, initially, equal to t_2. When an added edge of the augmented structure is visited, the value $t = t_2 - d + 1$ is computed where d is the depth of the node reached by such edge. If t is a position in the first half of the $2m$ characters, then t_1 is updated by changing its value to t. Else, the procedure stops and t_2 is decreased by 1. If t_2 is a position in the second half of the $2m$ characters then t_1 and t_2 are the first and last position of a boundary match, else there is no boundary match.

IV. The Greedy Approach

We provide a finite-state machine implementation of the greedy approach with an arbitrary dictionary. Then, we show the two implementations on standard large scale and extreme distributed systems.

A. The Finite-State Machine Implementation

We show the finite-state machine implementation producing the on-line greedy factorization of a string with an arbitrary dictionary. The most general formulation for a finite-state machine M is to define it as a six-tuple $M = (A, B, Q, \delta, q_0, F)$ with an input alphabet A, an output alphabet B, a set of states Q, a transition function $\delta : Q\mathrm{x}A \to Q\mathrm{x}B^*$, an initial state q_0 and a set of accepting states $F \subseteq Q$. The trie storing the dictionary is a subgraph of the finite-state machine diagram. It is well-known that each dictionary element is represented as a path from the root to a node of the trie where edges are labeled with an alphabet character (the root representing the empty string). The edges are directed from the parent to the child and the set of nodes represent the set of states of the machine. The output alphabet is binary and the factorization is represented by a binary string having the same length as the input string. The bits of the output string equal to 1 are those corresponding to the positions where the factors start. Since every string can be factorized, every state is accepting. The root represents the initial state. We need only to complete the function δ, by adding the missing edges of the diagram. The empty string is associated as output to the edges in the trie. For each node, the outgoing edges represent a subset of the input alphabet. Let f be the string (or dictionary element) corresponding to the node v in the trie and a an alphabet character not represented by an edge outgoing from v. Let $fa = f_1 \cdots f_k$ be the on-line greedy factorization of fa and i the smallest index such that $f_{i+1} \cdots f_k$ is represented by a node w in the trie. Then, we add to the trie a directed edge from v to w with label a. The output associated with the

edge is the binary string representing the sequence of factors $f_1 \cdots f_i$. By adding such edges, the machine is entirely defined. Redefining the machine to produce the compressed form of the string is straightforward.

B. The Distributed Implementations

In practice, greedy factorization is nearly optimal. As a first approach, we simply apply in parallel left to right greedy compression to blocks of length km. With standard scale systems, the block size must be the order of kilobytes to guarantee robustness. Each of the $O(n/km)$ processors could apply the finite-state machine implementation to its block. Beyond standard large scale, overlapping of adjacent blocks and a preprocessing of the boundaries are required as for the optimal case. Again, during the input phase overlapping blocks of length $m(k+2)$ are broadcasted to the processors as in the previous section. On the other hand, the definition of boundary match is extended to those factors, which are suffixes of the first half of the $2m$ characters shared by two adjacent blocks. The following procedure, even if it is not an approximation scheme from a theoretical point of view, performs in a nearly optimal way:

- for each block, every corresponding processor but the one associated with the last block computes the longest boundary match between its block and the next one;
- each processor computes the greedy factorization from the end of the boundary match on the left boundary of its block to the beginning of the boundary match on the right boundary.

The approach is nearly optimal for $k = 10$, as the approximation scheme of the previous section. The compression ratio achieved by such factorizarion is about the same as the sequential one. Considering that typically the average match length is 10, one processor can compress down to 100 bytes independently. This is why the approximation scheme was presented for massively parallel architecture and the approach, presented in this section, is suitable for extreme distributed systems, when the scale is beyond standard large values. Indeed, with a file size of several megabytes or more, the system scale has a greater order of magnitude than the standard large scale parameter. We wish to point out that the computation of the boundary matches is very relevant for the compression effectiveness, when an extreme distributed system is employed, since the sub-block length becomes much less than 1K. With standard large scale systems the block length is several kilobytes with just a few megabytes to compress and the approach using boundary matches is too conservative. After preprocessing, each of the $O(n/km)$ processors could apply the finite-state machine implementation to its block. However, blocks are so short that it becomes irrelevant. On the other hand, with standard scale systems and very large size files the application of the finite-state machine in parallel to the distributed blocks plays an important role to achieve high speed.

C. Speeding up the Preprocessing

To lower the time of the preprocessing phase to $O(m)$, the same augmented trie data structure, described in the previous section, is needed but, in this case, the boundary matches are the longest ones rather than the ones ending furthest to the right. Then, besides the temporary variables t_1 and t_2, employed by the preprocessing phase described in the previous section, two more variables τ_1 and τ_2 are required and, initially, equal to t_1 and t_2. Each time t_1 must be updated by such preprocessing phase, the value $t_2 - t_1 + 1$ is compared with $\tau_2 - \tau_1$ before updating. If it is greater or τ_2 is smaller than the last position of the first half of the $2m$ characters, τ_1 and τ_2 are set equal to t_1 and $t_2 - 1$. Then, t_1 is updated. At the end of the procedure, τ_1 and τ_2 are the first and last positions of the longest boundary match. We wish to point out that there is always a boundary match that is computed, since the final value of τ_2 always corresponds to a position equal either to one in the second half of the $2m$ characters or to the last position of the first half.

V. Experimental Results

Suffix-closed and prefix-closed dictionaries have been considered in static data compression because they are constructed by the LZ77 [24] and LZ78 [11] adaptive compression methods, when reading a typical string of a given source of data. When the input string to compress matches the characteristics of a dictionary given in advance and already filled with typical factors, the advantage in terms of compression efficiency is obvious. However, the bounded size of the dictionary (typically, 2^{16} factors) and its static nature imply a lack of robustness and the adaptive methods might result more effective in some cases, even if the type of data is known and the dictionary is very well constructed. We experimented this with the "compress" command line on the Unix and Linux platforms, which is the implementation of a variant of the LZ78 method, called the LZC method. LZC builds a prefix-closed dictionary of 2^{16} factors while compressing the data. When the dictionary is full, it applies static dictionary greedy compression monitoring at the same time the compression ratio. When the compression ratio starts deteriorating, it clears the dictionary and restarts dynamic compression alternating, in this way, adaptive and non-adaptive compression. We experimented that, when compressing megabytes of english text with a static prefix-closed dictionary optimally, there might be up to a ten percent loss in comparison with the compression ratio of the LZC method [23]. However, as we pointed out earlier, there is no scalable and robust implementation of the LZC method on a distributed memory system (except for the static phase of the method as shown in [26]), while a

nearly optimal compression distributed algorithm is possible with no scalability and robustness issues if we accept a ten percent compression ratio loss as a reasonable upper bound to the price to pay for it [23].

A prefix-closed dictionary D in [23] was filled up with 2^{16} elements, starting from the alphabet (each of the 256 bytes). Then, for each of the most common subbstrings listed in [9], every prefix of length less or equal to ten was added to D. On the other hand, for each string with no capital letters and less than eleven characters in the Unix dictionary of words, we added every prefix of length less or equal to six. For every word in the Unix dictionary inserted in D, a space was concatenated at the end of the copy in D. Another copy ending with the new line character was inserted if the word length is less than six. Finally, it was enough to add a portion of the words with six characters plus a new line character to fill up D. The average optimal compression ratio we obtained with this dictionary is 0.51, while the greedy one is even 0.57. On the other hand, the LZC average compression ratio is 0.42. It turned out that both gaps are consistently reduced when the prefix property of the dictionary is relaxed. A not prefix-closed dictionary D' was filled up with 2^{16} elements, starting from the alphabet and the 477 most common subbstrings listed in [9]. Then, we added each string with no capital letters and less than ten characters from the Unix dictionary of words. Again, for every word in the Unix dictionary inserted in D', a space was concatenated at the end of the copy in D'. Finally, it was enough to add a portion of short words with a new line character at the end to fill up D'. With such dictionary, the loss on the compression ratio goes down from ten to five percent with respect to the adaptive LZC compression. Moreover, the greedy approach has just a one percent loss with respect to optimal, as shown in Figure 4. This is because the dictionary is better constructed. In Figure 4, we also show the compression effectiveness results for the two approaches with or without boundaries preprocessing (that is, for an extreme or a standard distibuted system). The two approaches perform similarly and have a one percent loss with respect to sequential greedy, whether the dictionary is prefix-closed or not.

Dictionary	Optimal	Greedy	Standard	Extreme
Prefix-Closed	.51	.57	.58	.58
Not Prefix-Closed	.47	.48	.49	.49

Figure 4. Compression ratios with english text.

We observed in the introduction that for read-only memory files, speeding up decompression is what really matters in practice. In this context, the results presented in this paper suggest a dynamic approach (that is, working for any type of input), where the dictionary is not given in advance but learned from the input string and, then, used staticly to compress the string. This models a scheme where compression is performed only once with an off-line sequential procedure reading the string twice from left to right in such a way that decompression can be parallelized with no scalability issues. The first left-to-right reading is to learn the dictionary and better ways than the LZC algorithm exist since the dictionary provided by LZC, after reading the entire string, is constructed from a relatively short suffix of the input. A much more sophisticated approach employs the LRU (least recently used) strategy [9]. With such strategy, after the dictionay is filled up elements are removed in a continuous way by deleting at each step of the factorization the least recently used factor which is not a proper prefix of another one. A relaxed version of this approach was presented in [34], that is easier to implement, and experimental results show that the compression ratio with this type of dictionary goes down to 0.32 for english text [35]. This performance is kept if the greedy approach is applied staticly during the second reading of the string, using the dictionary obtained from the first reading. Moreover, if the compression is applied independently to different blocks of data of 1Kb or to smaller blocks after the boundaries preprocessing, there is still just a one percent loss on the compression ratio.

VI. Pseudo-Prefix and Pseudo-Suffix Dictionaries

We partially relax the suffix and prefix properties to keep respectively optimal the greedy and semi-greedy approaches by introducing pseudo-suffix and pseudo-prefix dictionaries. Then, we give an insight of why optimal factorizations can be computed in real time with pseudo-prefix and pseudo-suffix dictionaries while this is not possible if the dictionary is arbitrary. Finally, we present experimental results using pseudo-prefix dictionaries.

A. Introducing Pseudo-Prefix and -Suffix Dictionaries

Given a finite alphabet A, let p and s be a prefix and a suffix of a string $x \in A^*$ such that $x = ps$. Then, we call p the *complementary* prefix of s with respect to x. Accordingly, we call s the *complementary* suffix of p with respect to x. We say a dictionary D is *pseudo-prefix* if:

- let p be a prefix of $x \in D$ such that the complementary suffix s with respect to x is a prefix of an element in D. Then, $p \in D$.

Accordingly, we say a dictionary is *pseudo-suffix* if:

- let s be a suffix of $x \in D$ such that the complementary prefix p with respect to x is a suffix of an element in D. Then, $s \in D$.

We prove, now, the optimality of the on-line greedy factorization approach with pseudo-suffix dictionaries.

Theorem 1. Given a finite alphabet A, let $D \subseteq A^*$ be a pseudo-suffix dictionary. For every $x \in A^*$, the on-line greedy factorization of x is optimal.

Proof. The pseudo-suffix property implies, as the suffix property, that the on-line greedy approach selects, at each step, the factor ending furthest to the right. Indeed, assume that the factor selected by the greedy choice at the i-th step of the process ends to the right of the i-th factor of the optimal solution (which is always true at the first step). Then, there is a suffix s of the $i+1$-th factor of the optimal solution with a complementary prefix that is a suffix of the factor selected by the greedy choice at the i-th step. It follows that s is a dictionary element. Therefore, the on-line greedy approach selects, at each step, the factor ending furthest to the right and its optimality follows. q. e. d.

With the next theorem, we prove the optimality of the semi-greedy factorization process with pseudo-prefix dictionaries.

Theorem 2. Given a finite alphabet A, let $D \subseteq A^*$ be a pseudo-prefix dictionary. For every $x \in A^*$, the semi-greedy factorization of x is optimal.

Proof. The pseudo-prefix property implies, as the prefix property, that the semi-greedy approach selects, at each step, a factor such that the longest match in the next position with a dictionary element ends to the rightest. This is true at the first step, since for each suffix of the greedy factor that is a prefix of a dictionary element the complementary prefix is a dictionary element. Then, inductively, it is true for every step and the optimality follows. q. e. d.

It follows from the two theorems above and the results shown in the previous section that, as for prefix-closed and suffix-closed dictionaries, real-time optimal factorizations are possible with pseudo-prefix and pseudo-suffix dictionaries. Moreover, an optimal factorization using a pseudo-suffix dictionary is implementable with a finite state machine. However, the making of a pseudo-prefix dictionary is much simpler than the making of a pseudo-suffix one. Indeed, let D be an arbitrary dictionary stored in a trie and add prefixes of its elements to make it pseudo-prefix. The most natural way to do this is to visit the trie with a depth-first search. For each path from the root to a node representing a string not in D, such string is added to D if a descendant of the node is in D. The running time for such procedure is about the dictionary size times the depth of the trie.

B. Canonical Factors

We prove a property concerning optimal factorizations with pseudo-prefix and pseudo-suffix dictionaries. This property was previously proved for prefix-closed dictionaries in [18] and it gives an insight of why optimal factorizations can be computed in real time with this type of dictionary. First, we prove the property for pseudo-prefix dictionaries.

Theorem 3. Given a finite alphabet A, let $D \subseteq A^*$ be a pseudo-prefix dictionary. Let k be the number of factors of an optimal factorization of a string $s \in A^*$. Then, for $1 \leq i \leq k$, there is an optimal factorization such that its i-th factor is a substring of the i-th factor of every other optimal factorization of s.

Proof. First of all, given two optimal factorizations $s = f_1^1 \cdots f_k^1 = f_1^2 \cdots f_k^2$ we prove that f_i^1 and f_i^2 overlap for $1 \leq i \leq k$. Given any substring f of s, denote with $first(f)$ and $last(f)$ the first and the last position of s covered by f. Then, suppose $last(f_i^2) < first(f_i^1)$ for some i with $1 < 1 < k$. Let j be such that $first(f_j^1) \leq last(f_i^2) \leq last(f_j^1)$. It follows from the optimality of the two factorizations and the pseudo-prefix property that $last(f_j^1) < last(f_{i+1}^2)$. Denote with $pref(f_j^1)$ the prefix of f_j^1 such that $last(pref(f_j^1)) = last(f_i^2)$. Then, $pref(f_j^1) \in D$ since D is pseudo-prefix. It follows that the factorization $f_1^1 \cdots f_{j-1}^1 pref(f_j^1) f_{i+1}^2 \cdots f_k^2$ comprises less than k phrases since $j < i$. Therefore, f_i^1 and f_i^2 must have a not empty intersection. Suppose now that, for every optimal factorization $s = f_1 \cdots f_k$, $first(f_i) \leq first(f_i^1)$ and $last(f_i) \geq last(f_i^2)$. Denote with $pref(f_i^1)$ the prefix of f_i^1 such that $last(pref(f_i^1)) = last(f_i^2)$. Then, $pref(f_i^1) \in D$ since D is pseudo-prefix. It follows that $f_1^1 \cdots f_{i-1}^1 pref(f_i^1) f_{i+1}^2 \cdots f_k^2$ is an optimal factorization of s, with $pref(f_i^1)$ substring of the i-th factor of every other optimal factorization. q. e. d.

In the next theorem, we prove the property for pseudo-suffix dictionaries with similar arguments.

Theorem 4. Given a finite alphabet A, let $D \subseteq A^*$ be a pseudo-suffix dictionary. Let k be the number of factors of an optimal factorization of a string $s \in A^*$. Then, for $1 \leq i \leq k$, there is an optimal factorization such that its i-th factor is a substring of the i-th factor of every other optimal factorization of s.

Proof. First of all, given two optimal factorizations $s = f_1^1 \cdots f_k^1 = f_1^2 \cdots f_k^2$ we prove that f_i^1 and f_i^2 overlap for $1 \leq i \leq k$. Given any substring f of s, denote with $first(f)$ and $last(f)$ the first and the last position of s covered by f, as in Theorem 3. Then, suppose $last(f_i^2) < first(f_i^1)$ for some i with $1 < 1 < k$.

Let j be such that $first(f_j^1) \leq last(f_i^2) \leq last(f_j^1)$. It follows from the optimality of the two factorizations and the pseudo-suffix property that $first(f_j^1) > first(f_{i-1}^2)$. Denote with $suff(f_i^2)$ the suffix of f_i^2 such that $first(suff(f_i^2)) = first(f_j^1)$. Then, $suff(f_i^2) \in D$ since D is pseudo-suffix. It follows that the factorization $f_1^1 \cdots f_{j-1}^1 suff(f_i^2) f_{i+1}^2 \cdots f_k^2$ comprises less than k phrases since $j < i$. Therefore, f_i^1 and f_i^2 must have a not empty intersection. Suppose now that, for every optimal factorization $s = f_1 \cdots f_k$, $first(f_i \leq first(f_i^1)$ and $last(f_i) \geq last(f_i^2)$. Denote with $suff(f_i^2)$ the suffix of f_i^2 such that $first(suff(f_i^2)) = first(f_i^1)$. Then, $suff(f_i^2) \in D$ since D is pseudo-suffix. It follows that $f_1^1 \cdots f_{i-1}^1 suff(f_i^2) f_{i+1}^2 \cdots f_k^2$ is an optimal factorization of s, with $suff(f_i^2)$ substring of the i-th factor of every other optimal factorization. q. e. d.

Given a finite alphabet A, a dictionary $D \subseteq A^*$ and a string $s \in A^*$, let a string $f \in A^*$ be the i-th factor of an optimal factorization of s with respect to D for some positive integer i less or equal to the optimal cost k. Then, we call f *canonical* if it is the substring of the i-th of every other optimal factorization of s. We proved, in the two theorems above, that if the dictionary is pseudo-prefix or pseudo-suffix then, given any input string, for every positive integer between 1 and the optimal factorization cost there is a canonical factor. The presence of these cannical factors gives an insight of why a real-time factorization is possible for this type of dictionaries since it proves that in order to determine the next factor of an optimal factorization we need to process only the current one.

Now, we show the impossibility of a real-time optimal factorization for every input string if the dictionary is arbitrary, by presenting an example where the dictionary is $\{a, b, a^i, a^j, a^{k(j-i)}b^2\}$ with $ki < j$. Then, we consider two strings s_1 and s_2 sharing the same prefix a^{kj} but with two different complementary suffixes equal, respectively, to $a^{k(j-i)}b^2$ and b^2. Then, the optimal factorization is $a^j, ..., a^j, a^{k(j-i)}b^2$ for s_1 and $a^i, ..., a^i, a^{k(j-i)}b^2$ for s_2. This proves that any approach to produce an optimal factorization is not independent from the maximum factor length L of the dictionary and that the complexity of the optimal factorization problem is $\Omega(nL)$, where n is the input string length.

C. Experimental Results

The results presented in Figure 4 for a not prefix-closed dictionary D' are reported again in Figure 5 as results for a not pseudo-prefix dictionary, since the dictionary D' described in the previous section was not pseudo-prefix as well. If we add prefixes to D' to make it pseudo-prefix, optimal compression is the same as before and greedy is basically optimal (less than one percent loss), as shown in Figure 5. We also show that the compression effectiveness results for the pseudo-prefix dictionary with or without boundaries preprocessing (that is, for an extreme or a standard distibuted system) have a one percent loss with respect to sequential greedy, so the pseudo-prefix dictionary has a better performance.

Dictionary	Optimal	Greedy	Standard	Extreme
Pseudo-Prefix	.47	.47	.48	.48
Not Pseudo-Prefix	.47	.48	.49	.49

Figure 5. Compression ratios with english text.

Now, we consider the off-line dynamic approach reading twice the input, which can be applied in the case of read-only memory files. The dictionary, bounded by the LRU strategy, is not pseudo-prefix if it is learned by the heuristic in [13]. We mentioned in the previous section the average compression ratio for english text is 0.32 and if the compression is applied independently to different blocks of data of 1Kb or to smaller blocks after the boundaries preprocessing, there is still just a one percent loss on the compression ratio. This loss disappears if we make the dictionary pseudo-prefix and apply the approximation scheme in [23] to optimal compression.

VII. Conclusion

We presented parallel implementations of the greedy approach to dictionary-based static text compression suitable for standard and non-standard large scale distributed systems. In order to push scalability beyond what is traditionally considered a large scale system, a more involved approach distributes overlapping blocks to compute boundary matches. These boundary matches are relevant to maintain the compression effectiveness on a so-called extreme distributed system. If we have a standard small, medium or large scale system available, the approach with no boundary matches can be used. The absence of a communication cost during the computation guarantees a linear speed-up. Moreover, the finite state machine implementation speeds up the execution of the distributed algorithm in a relevant way when the data blocks are large, that is, when the size of the input file is large and the size of the distributed system is relatively small. Finally, we introduced the notion of pseudo-prefix dictionary, which allows optimal compression by means of a real-time semi-greedy procedure and a slight improvement on the compression ratio obtained by the distributed implementations. As future work, experiments on parallel running times should be done to see how the preprocessing phase effects on the linear speed-up when the system is scaled up beyond the standard size and how relevant the employment of the finite state machine implementation is when the data blocks are small.

REFERENCES

[1] S. DeAgostino, "The Greedy Approach to Dictionary-Based Static Text Compression on a Distributed System," Proceedings International Conference on Advances Engeneering Computing with Applications to Sciences, 2014, pp. 1-6.

[2] S. DeAgostino, "Approximating Dictionary-Based Optimal Data Compression on a Distributed System," to appear in Proceedings ACM International Conference on Computing Frontiers, 2015.

[3] R. A. Wagner, "Common Phrases and Minimum Text Storage," Communications of the ACM, vol. 16, 1973, pp. 148-152.

[4] E. J. Shoegraf and H. S. Heaps, "A Comparison of Algorithms for Data Base Compression by Use of Fragments as Language Elements," Information Storage and Retrieval, vol. 10, 1974, pp. 309-319.

[5] M. Cohn and R. Khazan, "Parsing with Suffix and Prefix Dictionaries," Proceedings IEEE Data Compression Conference, 1996, pp. 180-189.

[6] M. Crochemore and W. Rytter, Jewels of Stringology, World Scientific, 2003.

[7] A Hartman and M. Rodeh, "Optimal Parsing of Strings," Combinatorial Algorithms on Words (eds. Apostolico, A., Galil, Z.), Springer, 1985, pp. 155-167.

[8] T. C. Bell, J. G. Cleary, and I. H. Witten, Text Compression, Prentice Hall, 1990.

[9] J. A. Storer, Data Compression: Methods and Theory, Computer Science Press, 1988.

[10] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences," IEEE Transactions on Information Theory, vol. 22, 1976, pp. 75-81.

[11] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," IEEE Transactions on Information Theory, vol. 24, 1978, pp. 530-536.

[12] T. A. Welch, "A Technique for High-Performance Data Compression," IEEE Computer, vol. 17, 1984, pp. 8-19.

[13] V. S. Miller and M. N. Wegman, "Variations on Theme by Ziv - Lempel," Combinatorial Algorithms on Words (eds. Apostolico, A., Galil, Z.), Springer, 1985, pp. 131-140.

[14] L. Cinque, S. De Agostino, and L. Lombardi, "Scalability and Communication in Parallel Low-Complexity Lossless Compression," Mathematics in Computer Science, vol. 3, 2010, pp. 391-406.

[15] S. De Agostino, Sub-Linear Algorithms and Complexity Issues for Lossless Data Compression, Master's Thesis, Brandeis University, 1994.

[16] S. De Agostino, Parallelism and Data Compression via Textual Substitution, Ph. D. Dissertation, Sapienza University of Rome, 1995.

[17] S. De Agostino, "Parallelism and Dictionary-Based Data Compression," Information Sciences, vol. 135, 2001, pp. 43-56.

[18] S. De Agostino S. and J. A. Storer, "Parallel Algorithms for Optimal Compression Using Dictionaries with the Prefix Property," Proceedings IEEE Data Compression Conference, 1992, pp. 52-61.

[19] D. S. Hirschberg and L. M. Stauffer, "Parsing Algorithms for Dictionary Compression on the PRAM," Proceedings IEEE Data Compression Conference, 1994, pp. 136-145.

[20] D. S. Hirschberg and L. M. Stauffer, "Dictionary Compression on the PRAM," Parallel Processing Letters, vol. 7, 1997, pp. 297-308.

[21] H. Nagumo, M. Lu, and K. Watson, "Parallel Algorithms for the Static Dictionary Compression," Proceedings IEEE Data Compression Conference, 1995, pp. 162-171.

[22] L. M. Stauffer and D. S. Hirschberg, "PRAM Algorithms for Static Dictionary Compression," Proceedings International Symposium on Parallel Processing, 1994, pp. 344-348.

[23] D. Belinskaya, S. De Agostino, and J. A. Storer, "Near Optimal Compression with respect to a Static Dictionary on a Practical Massively Parallel Architecture," Proceedings IEEE Data Compression Conference, 1995, pp. 172-181.

[24] A. Lempel and J. Ziv, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, vol. 23, 1977, pp. 337-343.

[25] S. DeAgostino, "Parallel Implementations of Dictionary Text Compression without Communication," London Stringology Days, 2009.

[26] S. DeAgostino, "LZW Data Compression on Large Scale and Extreme Distributed System," Proceedings Prague Stringology Conference, 2012, pp. 18-27.

[27] Y. Matias and C. S. Sahinalp, "On the Optimality of Parsing in Dynamic Dictionary-Based Data Compression," Proceedings SIAM-ACM Symposium on Discrete Algorithms, 1999, pp. 943-944.

[28] M. Crochemore, A. Langiu, and F. Mignosi, "Note on the Greedy Parsing Optimality for Dictionary-Based Text Compression," Theoretical Computer Science, vol. 525, 2014, pp. 55-59.

[29] M. Crochemore, L. Gianbruno, A. Langiu, F. Mignosi, and A. Restivo, "Dictionary-Simbolwise Flexible Parsing," Journal of Discrete Algorithms, vol. 14, 2012, pp. 74-90.

[30] A. Farrugia, P. Ferragina, A. Frangioni, and R. Venturini, "Bicriteria Data Compression," Proceedings SIAM-ACM Symposium on Discrete Algorithms, 2014, pp. 1582-1585.

[31] P. Ferragina, I. Nitto, and R. Venturini, "On Optimally Partitioning a Text to Improve Its Compression," Algorithmica, vol. 61, 2011, pp. 51-74.

[32] P. Ferragina, I. Nitto, and R. Venturini, "On the Bit-Complexity of Lempel-Ziv Compression," SIAM Journal on Computing, vol. 42, 2013, pp. 1521-1541.

[33] A. Langiu, "On Parsing Optimality for Dictionary-Based Text Compression - the Zip Case", Journal of Discrete Algorithms, vol. 20, 2013, pp. 65-70.

[34] S. DeAgostino and R. Silvestri, "Bounded Size Dictionary Compression: SC^k-Completeness and NC Algorithms," Information and Computation, vol. 180, 2003, pp. 101-112.

[35] S. DeAgostino, "Bounded Size Dictionary Compression: Relaxing the LRU Deletion Heuristic," Proceedings Prague Stringology Conference, 2005, pp. 135-142.

τOWL: A Framework for Managing Temporal Semantic Web Documents Supporting Temporal Schema Versioning

Abir Zekri
University of Sfax
Sfax, Tunisia
abir.zekri@fsegs.rnu.tn

Zouhaier Brahmia
University of Sfax
Sfax, Tunisia
zouhaier.brahmia@fsegs.rnu.tn

Fabio Grandi
University of Bologna
Bologna, Italy
fabio.grandi@unibo.it

Rafik Bouaziz
University of Sfax
Sfax, Tunisia
raf.bouaziz@fsegs.rnu.tn

***Abstract* — The OWL 2 Web Ontology Language allows defining both schema and instances of ontologies for Semantic Web applications, but lacks explicit support for time-varying ontologies. Hence, knowledge engineers or maintainers of Semantic Web documents have to use *ad hoc* techniques in order to specify an OWL 2 schema for time-varying instances and to cope with its temporal evolution. In this paper, for a disciplined and systematic approach to the temporal management of OWL 2 ontologies, we propose the adoption of a framework called Temporal OWL 2 (τOWL), inspired by the Temporal XML Schema (τXSchema) framework defined for XML data. Hence, τOWL allows creating a temporal OWL 2 ontology from a conventional (i.e., non-temporal) OWL 2 ontology and a set of logical and physical annotations. Logical annotations identify which elements of a Semantic Web document can vary over time and physical annotations specify how the time-varying aspects are represented in the document. By using annotations to integrate temporal aspects in the traditional Semantic Web, our framework (i) guarantees logical and physical data independence for temporal schemas and (ii) provides a low-impact solution since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards. Furthermore, temporal versioning of the schema itself is supported in τOWL by means of a temporal schema, which is a document that binds the three components of a τOWL schema to the temporal versions they belong to. In τOWL, either the conventional schema and the temporal schema can be versioned, by means of two dedicated complete sets of schema change primitives, which are defined in this work. We also illustrate their use and show their impact on OWL 2 instances through an example.**

Keywords – Semantic Web; Ontology; OWL 2; τXSchema; Logical annotations; Physical annotations; Temporal database; XML Schema; XML; τOWL; Conventional schema; Temporal schema; Schema versioning; Temporal ontology; Ontology versioning

I. INTRODUCTION

Time is an omnipresent dimension in both modern [1] and classical [2] applications; it is used to timestamp data values to keep track of changes in the real world and model their history. Hence, studying time has been, and continues to be, one of the main research interests in different scientific fields, such as databases and knowledge representation.

Since the second half of the 1980s, a great deal of work has been done in the field of temporal databases [3][4][5]. Several data models and query languages have been proposed for the management of time-varying data. Temporal databases usually adopt one or two time dimensions to timestamp data: (a) transaction time, which indicates when an event is recorded in the database, and (b) valid time, which represents the time when an event occurred, occurs or is expected to occur in the real world. Bi-temporal data are timestamped by both transaction time and valid time dimensions. Snapshot data are traditional data, without time support.

On the other hand, the World Wide Web (WWW or Web) [6] was shifted from the semi-structured internet to a more structured Web called the Semantic Web [7][8]. The new generation of Web aims at providing languages and tools that specify explicit semantics for data and enable knowledge sharing among knowledge-based applications. In this vision, ontologies [9] are used for defining and relating concepts that describe Web resources, in a formal way. The new emerging standard for describing ontologies, which has been recommended by the W3C since 2009, is OWL 2 [10][11][12]. It allows defining both schema (in terms of entities, axioms, and expressions) and instances (i.e., individuals) of ontologies; OWL 2 ontologies are stored as Semantic Web documents.

Due to the dynamic nature of the Web, ontologies that are used on the Web (like other Web application components such as Web databases, Web pages and Web scripts) evolve over time to reflect and model changes occurring in the real-world. Furthermore, several Semantic Web-based applications (like e-commerce, e-government and e-health applications) require keeping track of ontology evolution and versioning with respect to time, in order to represent, store and retrieve time-varying ontologies.

Unfortunately, while there is a sustained interest for temporal and evolution aspects in the research community [13], existing Semantic Web standards but also state-of-the-art ontology editors and knowledge representation tools do not provide any built-in support for managing temporal ontologies. In particular, the W3C OWL 2 recommendation lacks explicit support for time-varying ontologies, at both schema and instance levels. Thus, knowledge engineers or maintainers of semantics-based Web resources must use *ad*

hoc techniques when there is a need, for example, to specify an OWL 2 ontology schema for time-varying ontology instances or to deal with temporal evolution of the ontology schema itself. In the rest of the paper, we define as Knowledge Base Administrator (KBA) a knowledge engineer or, more in general, the person in charge of the maintenance of semantics-based Web resources.

According to what precedes, we think that if we would like to handle ontology evolution over time in an efficient manner and to allow historical queries to be executed on time-varying ontologies, a built-in temporal ontology management system is needed. For that purpose, we propose in this paper a framework, called τOWL, for managing temporal Semantic Web documents, through the use of a temporal OWL 2 extension. In fact, we want to introduce with τOWL a principled and systematic approach to the temporal extension of OWL 2, similar to that Snodgrass and colleagues did to the XML language with τXSchema [14][15][16]. τXSchema is a framework (i.e., a data model equipped with a suite of tools) for managing temporal XML documents, well known in the database research community and, in particular, in the field of temporal XML [17]. Moreover, in our previous work [18][19][20], with the aim of completing the framework, we augmented τXSchema by defining necessary schema change operations acting on conventional schema, temporal schema, and logical and physical annotations (extensions which we plan to apply to τOWL too).

Being defined as a τXSchema-like framework, τOWL facilitates the creation of a temporal OWL 2 ontology from a conventional (i.e., non-temporal) OWL 2 ontology specification and a set of logical (or temporal) and physical annotations. Logical annotations identify which components of a Semantic Web document can vary over time; physical annotations specify how the time-varying aspects are represented in the document. By using temporal schema and annotations to introduce temporal aspects in the conventional (i.e., non temporal) Semantic Web, our framework (i) guarantees logical and physical data independence [21] for temporal schemas and (ii) provides a low-impact solution since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards.

Furthermore, with respect to the preliminary version of this work presented at SEMAPRO 2014 [1], in this paper we extend the τOWL framework to also support schema versioning [22][23], which is the most powerful technique for managing the history of schema changes. Since ontology schemata are also evolving over time to reflect changes in real-world applications [24], keeping a fully fledged history of ontology changes (i.e., involving both the ontology instances and the ontology schema) is a very required feature for many Semantic Web applications. More precisely, we present our technique for the versioning of a τOWL schema, and define necessary schema change operations acting on conventional ontology schema and on temporal ontology schema. We do not deal in this paper with changes acting on logical and physical annotations; that will be studied in a future work.

The remainder of the paper is organized as follows. Section II motivates the need for an efficient management of time-varying Semantic Web documents. Section III describes the τOWL framework that we propose for extending the Semantic Web to temporal aspects: the architecture of τOWL is presented and details on all its components and support tools are given. Section IV presents our approach for versioning of a τOWL schema. Section V introduces the schema change primitives that we propose, in the τOWL framework, for changing the conventional schema and for updating the temporal schema. Section VI discusses related work. Section VII provides a summary of the paper and some remarks about our future work.

II. MOTIVATION

In this section, we present a motivating example that shows the limitation of the OWL 2 language for explicitly supporting time-varying instances. Then, we state the desiderata for an OWL 2 extension, which could accommodate time-varying instances in a disciplined and systematic way.

A. Running Example

As a motivating and illustrative example for τOWL, we recall and extend the example presented in the preliminary version of this work [1], dealing with the management of the evolution of an ontology based on Friend Of A Friend (FOAF). The FOAF project [25] is creating a Web of machine-readable pages describing people, the links between them and the things they create and do.

Suppose that the Web site "Web-S1" publishes the FOAF definition for his user "Nouredine". A fragment of the FOAF Resource Description Framework (RDF [26]) document of "Nouredine" is presented in Figure 1. It describes, according to the FOAF ontology, the personal information of "Nouredine" (i.e., name and nickname) and the information about his online accounts on diverse sites (i.e., the home page of the site, and the account name of the user). In this example, we limit to describe user's information concerning the account on the online Web site "Facebook".

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <foaf:nick>Nor</foaf:nick>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <foaf:accountName>Nor_Tunsi
      </foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 1. A fragment of Nouredine FOAF RDF document on January 15, 2014.

Assume that information about the user "Nouredine" of the Web site "Web-S1" was added on January 15, 2014. On February 08, 2014, Nouredine modified his nickname from

"Nor" to "Nouri" and his account name of Facebook from "Nor_Tunsi" to "Nouri_Tunsi". Thus, the corresponding fragment of the Nouredine FOAF RDF document was revised to that shown in Figure 2.

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <foaf:nick>Nouri</foaf:nick>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <foaf:accountName>Nouri_Tunsi
      </foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
...
```

Figure 2. A fragment of Nouredine FOAF RDF document on February 08, 2014.

In many Semantic Web-based applications, the history of ontology changes is a fundamental requirement, since such a history allows recovering past ontology versions, tracking changes over time, and evaluating temporal queries [27]. A τOWL time-varying Semantic Web document records the evolution of a Semantic Web document over time by storing all versions of the document in a way similar to that originally proposed for τXSchema [14].

Suppose that the webmaster of the Web site "Web-S1" would like to keep track of the changes performed on our FOAF RDF information by storing both versions of Figure 1 and of Figure 2 in a single (temporal) RDF document. As a result, Figure 3 shows a fragment of a time-varying Semantic Web document that captures the history of the specified information concerning "Nouredine".

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <versionedNick>
    <NickVersion>
      <nickValidityStartTime>2014-01-15
      </nickValidityStartTime>
      <nickValidityEndTime>2014-02-07
      </nickValidityEndTime>
      <foaf:nick>Nor</foaf:nick>
    </NickVersion>
    <NickVersion>
      <nickValidityStartTime>2014-02-08
      </nickValidityStartTime>
      <nickValidityEndTime>now
      </nickValidityEndTime>
      <foaf:nick>Nouri</foaf:nick>
    </NickVersion>
  </versionedNick>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <versionedAccountName>
        <AccountNameVersion>
          <accountNameValidityStartTime>
              2014-01-15
          </accountNameValidityStartTime>
          <accountNameValidityEndTime>
              2014-02-07
          </accountNameValidityEndTime>
          <foaf:accountName>Nor_Tunsi
          </foaf:accountName>
        </AccountNameVersion>
        <AccountNameVersion>
          <accountNameValidityStartTime>
              2014-02-08
          </accountNameValidityStartTime>
          <accountNameValidityEndTime>
              now
          </accountNameValidityEndTime>
          <foaf:accountName>Nouri_Tunsi
          </foaf:accountName>
        </AccountNameVersion>
      </versionedAccountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
...
```

Figure 3. A fragment of the time-varying Nouredine FOAF RDF document.

In this example, we use valid-time to capture the history of such information. In order to timestamp the entities which can evolve over time, we use the following optional tags: **nickValidityStartTime** and **nickValidityEndTime**, for recording **nick** name evolution, and **accountNameValidityStartTime** and **accountNameValidityEndTime**, for keeping the **accountName** history. These are optional Data Properties which can be added to a temporal entity. The domain of nickValidityEndTime or accountNameValidityEndTime includes the value "now" [28]; the entity that has "now" as the value of its validity end time property represents the current entity until some change occurs.

Assume that the extract of the FOAF ontology presented in Figure 4 contains the conventional (i.e., non-temporal) schema [14] for the FOAF RDF document presented in both Figure 1 and Figure 2. The conventional schema is the schema for an individual version, which allows updating and querying individual versions.

```
<rdf:RDF>
  <owl:Ontology  rdf:about="http://purl.org/
                            az/foaf#">
    <rdfs:Class  rdf:about="#Person">
      <rdf:type  rdf:resource="http://www.w3.org/
                               2002/07/owl#Class"/>
    </rdfs:Class>
    <rdf:Property  rdf:about="#holdsAccount">
      <rdf:type  rdf:resource="http://www.w3.org/
                       2002/07/owl#ObjectProperty"/>
      <rdfs:domain  rdf:resource="#Person"/>
      <rdfs:range  rdf:resource="#OnlineAccount"/>
    </rdf:Property>
    <rdf:Property  rdf:about="#accountName">
      <rdf:type  rdf:resource="http://www.w3.org/
                     2002/07/owl#DatatypeProperty"/>
      <rdfs:domain  rdf:resource="#OnlineAccount"/>
    </rdf:Property>
    …
</rdf:RDF>
```

Figure 4. An RDF/XML extract from the OWL 2 FOAF ontology.

The problem is that the time-varying ontology document

(see Figure 3) does not conform to the conventional ontology schema (see Figure 4). Thus, to resolve this problem, we need a different ontology schema that can describe the structure of the time-varying ontology document. This new schema should specify, for example, timestamps associated to entities, time dimensions involved, and how the entities vary over time. This example will be continued in Section III.A, in order to show how these problems can be solved in our proposed τOWL framework.

Furthermore, we want our framework also allows KBAs to effect and keep track of changes to the conventional schema itself. In Section V.D, we will complete this example by describing some changes made by the KBA on this initial framework and showing their effects both at schema and at instance levels.

B. Desiderata

There are several goals that can be fulfilled when augmenting the OWL 2 language to support time-varying instances. Our approach aims at satisfying the following requirements:

- facilitating the management of time for KBAs;
- supporting both valid time and transaction time;
- supporting (temporal) versioning of OWL 2 ontology instances;
- keeping compatibility with existing OWL 2 W3C recommendations, standards, and editors, without requiring any changes to these recommendations, standards, and tools;
- supporting existing applications that are already using OWL 2 ontologies;
- providing OWL 2 data independence so that changes at the logical level are isolated from those performed at the physical level, and vice versa;
- accommodating a variety of physical representations for time-varying OWL 2 instances;
- supporting (temporal) versioning of OWL 2 ontology schemata.

III. THE τOWL FRAMEWORK

In this section, we present our τOWL framework for handling temporal Semantic Web documents and provide an illustrative example of its use. We describe the overall architecture of τOWL and the tools used for managing both τOWL schema and τOWL instances. Since τOWL is a τXSchema-like framework, we were inspired by the τXSchema architecture and tools while defining the architecture and tools of τOWL.

The τOWL framework allows a KBA to create a temporal OWL 2 schema for temporal OWL 2 instances from a conventional OWL 2 schema, logical annotations, and physical annotations. Since it is a τXSchema-like framework, τOWL use the following principles:

- separation between (i) the conventional (i.e., non-temporal) schema and the temporal schema, and (ii) the conventional instances and the temporal instances;
- use of temporal and physical annotations to specify temporal and physical aspects, respectively, at schema level.

Figure 5 illustrates the architecture of τOWL. Notice that only the components that are presented in the figure as rectangular pink boxes with bold border are specific to an individual time-varying OWL 2 document and need to be supplied by a KBA. The framework is based on the OWL 2 language [10], which is a W3C standard ontology language for the Semantic Web. It allows defining both schema (i.e., entities, axioms, and expressions) and instances (i.e., individuals) of ontologies. Thus, we consider that the signature of an OWL 2 ontology O can be defined as follows: O = {E, A, Exp} such that:

i) E = {C, DP, OP, AP} represents the set of the entities with:
   - C: Class, represents the set of concepts;
   - DP: Data Property, represents the set of properties of the concepts;
   - OP: Object Property, represents the set of the semantic relations between the concepts;
   - AP: Annotation Property, represents the set of annotations on the entities and those on the axioms.

ii) A = {EAx, KAx} represents the set of axioms with:
   - EAx: Entity Axioms, represents the axioms which concern the entities;
   - KAx: Key Axioms, represents all the identifiers associated to the various classes.

iii) Exp = {CE, OPE, DPE} represents the set of the used expressions (an expression is a complex description which results from combinations of entities by using constructors such as enumeration, restriction of cardinality and restriction of properties) with:
   - CE: Class Expressions, represents the set of combinations of concepts by using constructors;
   - OPE: Object Property Expressions, represents the set of combinations of relations;
   - DPE: Data Property Expressions, represents the set of combinations of properties.

The KBA starts by creating the *conventional schema* (box 7), which is an OWL 2 ontology that models the concepts of a particular domain and the relations between these concepts, without any temporal aspect. To each conventional schema corresponds a set of conventional (i.e., non-temporal) OWL 2 instances (box 12). Any change to the conventional schema is propagated to its corresponding instances. Notice that our approach deals with OWL 2 ontologies with an RDF/XML syntax [29], which is, according to the OWL 2 specification document [11], the only syntax that must mandatorily be supported by OWL 2 tools.

After that, the KBA augments the conventional schema with *logical* and *physical annotations*, which allow him/her to express, in an explicit way, all requirements dealing with the representation and the management of temporal aspects associated to the components of the conventional schema, as described in the following.

Logical annotations [16] allow the KBA to specify:

1) whether a conventional schema component varies over valid time and/or transaction time;

2) whether its lifetime is described as a continuous state or a single event;

3) whether the component may appear at certain times (and not at others);

4) whether its content changes.

If no logical annotations are provided, the default logical annotation is that anything can change. However, once the conventional schema is annotated, components that are not described as time-varying are static and, thus, they must have the same value across every instance document (box 12).

Physical annotations [16] allow the KBA to specify the timestamp representation options chosen, such as where the timestamps are placed and their kind (i.e., valid time or transaction time) and the kind of representation adopted. The location of timestamps is largely independent of which components vary over time. Timestamps can be located either on time-varying components (as specified by the logical annotations) or somewhere above such components. Two OWL 2 documents with the same logical information will look very different if we change the location of their physical timestamps. Changing an aspect of even one timestamp can make a big difference in the representation. τOWL supplies a default set of physical annotations, which is to timestamp the root element with valid and transaction times. However, explicitly defining them can lead to more compact representations [16].

In order to improve conceptual clarity and also to enable a more efficient implementation, we adopt a "separation of concerns" principle in our approach: since the entities, the axioms and the expressions of an OWL 2 ontology evolve over time independently, we distinguish between three separate types of annotations to be defined and to be associated to a conventional schema: the *entity annotations* (box 9), the *axiom annotations* (box 10) and the *expression annotations* (box 11).

Entity annotations describe the logical and physical characteristics associated to the components of an OWL 2 ontology: classes, relations, and properties. They indicate for example the temporal formats of these components, which could be valid-time, transaction-time, bi-temporal or snapshot (by default). The schema for the logical and physical entity annotations is given by *EntASchema* (box 4). Axiom annotations and expression annotations describe the logical and physical aspects of axioms and expressions defined on classes or on properties. The schema for the logical and physical axiom annotations is given by *AxiASchema* (box 5) and the schema for the logical and physical expression annotations is given by *ExpASchema* (box 6).

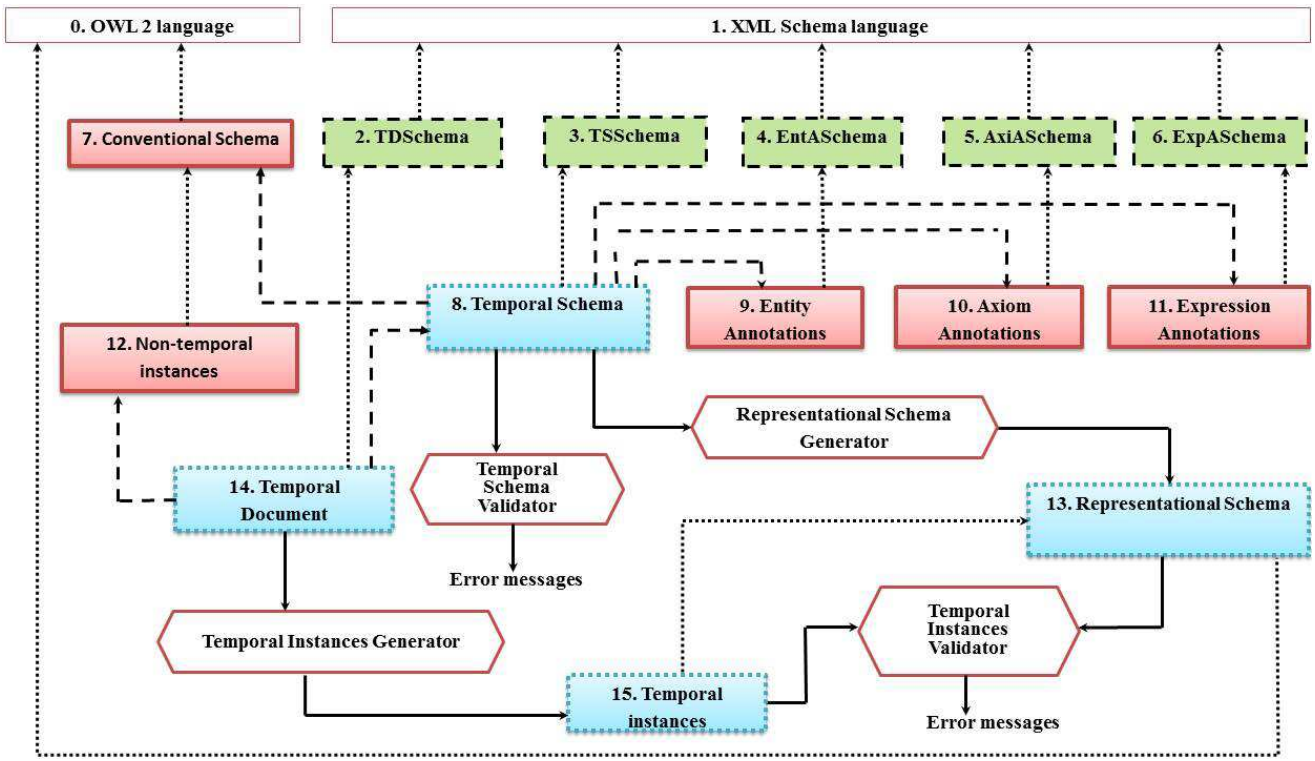


Figure 5. Overall architecture of τOWL. In the picture, rectangular boxes represent documents, hexagonal boxes represent tools, solid arrows denote Input/Output data flows, dotted arrows link documents to namespaces and dashed arrows stand for "references" relationships. Moreover, the meaning of the color and the border pattern of rectangular boxes is as follows: pink box with bold border for documents created/added by the KBA (7, 9, 10, 11 and 12), blue box with dotted border for documents automatically generated by the system (8, 13, 14, and 15), green box with dashed border for predefined documents making part of the framework (2, 3, 4, 5 and 6), and white box with thin border for reference documents created by the W3C (0 and 1).

Notice that EntASchema, AxiASchema, and ExpASchema, which all contain both logical and physical annotations, are XML Schemas [30]. The annotations associated to the same conventional schema can evolve independently. Any change to one of the three sets of annotations does not affect the two other sets.

Finally, when the KBA finishes annotating the conventional schema and asks the system to save his/her work, this latter creates the *temporal schema* (box 8) in order to provide the linking information between the conventional schema and its corresponding logical and physical annotations. The temporal schema is a standard XML document, which ties the conventional schema, the entity annotations, the axiom annotations, and the expression annotations together. In the τOWL framework, the temporal schema is the logical equivalent of the conventional OWL 2 schema in a non-temporal context. This document contains sub-elements that associate a series of conventional schema definitions with entity annotations, axiom annotations, and expression annotations, along with the time span during which the association was in effect. The schema for the temporal schema document is the XML Schema Definition document *TSSchema* (box 3).

To complete the figure in our temporal context, after creating the temporal schema, the system creates a *temporal document* (box 14) in order to link each conventional ontology instance document (box 12), which is valid to a conventional ontology schema (box 7), to its corresponding temporal ontology schema (box 8), and more precisely to its corresponding logical and physical annotations (which are referenced by the temporal schema). A temporal document is a standard XML document that maintains the evolution of a non-temporal ontology instance document over time, by recording all of the versions (or temporal slices) of the document with their corresponding timestamps and by specifying the temporal schema associated to these versions. This document contains sub-elements that associate a series of conventional ontology instance documents with logical and physical annotations (on entities, axioms, and expressions), along with the time span during which the association was in effect. Thus, the temporal document is very important for making easy the support of temporal queries working on past versions or dealing with changes between versions. The schema for the temporal document is the XML Schema Definition document *TDSchema* (box 2).

Notice that, whereas TDSchema (box 2), TSSchema (box 3), EntASchema (box 4), AxiASchema (box 5), and ExpASchema (box 6) have been developed by us, OWL 2 (box 0) and XML Schema (box 1) correspond to the standards endorsed by the W3C.

In a similar way to what happens in the τXSchema framework, the temporal schema document (box 8) is processed by the ***temporal schema validator*** tool in order to ensure that the logical and physical entity annotations, axiom annotations and expression annotations are (i) valid with respect to their corresponding schemas (i.e., EntASchema, AxiASchema, and ExpASchema, respectively), and (ii) consistent with the conventional schema. The temporal schema validator tool reports whether the temporal schema document is valid or invalid.

Once all the annotations are found to be consistent, the ***representational schema generator*** tool generates the *representational schema* (box 13) from the temporal schema (i.e., from the conventional schema and the logical and physical annotations); it is the result of transforming the conventional schema according to the requirements expressed through the different annotations. The representational schema becomes the schema for temporal instances (box 15). Temporal instances could be created in four ways:

i) automatically from the temporal document (box 14) (i.e., from *non-temporal ontology instances* (box 12) and the temporal ontology schema (box 8)), using the ***temporal instances generator*** tool (such an operation is called "squash" in the original τXSchema approach);
ii) automatically from instances stored in a knowledge base, i.e., as the result of a "temporal query" or a "temporal view";
iii) automatically from a third-party tool, or
iv) manually (i.e., temporal instances are directly inserted by the KBA into the τOWL repository).

Moreover, temporal instances are validated against the representational schema through the ***temporal instances validator*** tool, which reports whether the temporal instances document (box 15) is valid or invalid.

Notice that the four mentioned tools (i.e., Temporal Schema Validator, Temporal Instances Validator, Representational Schema Generator, and Temporal Instances Generator) are under development. For example, the temporal instances validator tool is being implemented as a temporal extension of an existing conventional ontology instance validator.

A. Running example reprise

In order to show the functioning of the proposed approach, we continue in the following our motivating example of Section II.A, in order to show how management of temporal ontology document versions is dealt with in the τOWL approach.

On January 15, 2014, the KBA creates a conventional ontology schema (box 7), named "PersonSchema_V1.owl" (as in Figure 4), and a conventional ontology document (box 12), named "Persons_V1.rdf" (as in Figure 1), which is valid with respect to this schema. We assume that the KBA defines also a set of logical and physical annotations, associated to that conventional schema; they are stored in an ontology annotation document (boxes 9, 10, and 11) titled "PersonAnnotations_V1.xml" as shown in Figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<ontologyAnnotationSet>
  <logicalAnnotations>
    <item target="/Person/nick">
      <validTime kind="state"
                 content="varying"
                 existence="constant"/>
    </item>
  </logicalAnnotations>
  <physicalAnnotations>
```

```
    <stamp target="Person/nick"
           dataInclusion="expandedVersion">
      <stampkind timeDimension="validTime"
                 stampBounds="extent"/>
    </stamp>
  </physicalAnnotations>
</ontologyAnnotationSet>
```

Figure 6. The annotation document on January 15, 2014.

After that, the system creates the temporal ontology schema (box 8) in Figure 7, which ties "PersonSchema_V1.owl" and "PersonAnnotations_V1.xml" together; this temporal schema is saved in an XML file titled "PersonTemporalSchema.xml". Consequently, the system uses the temporal ontology schema of Figure 7 and the conventional ontology document in Figure 1 to create a temporal document (box 14) as in Figure 8, which lists both versions (i.e., temporal "slices") of the conventional ontology documents with their associated timestamps. The squashed version (box 15) of this temporal document, which could be generated by the Temporal Instances Generator, is provided in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location="PersonSchema_V1.owl"
             begin="2014-01-15" />
    </sliceSequence>
  </conventionalOntologySchema>
  <ontologyAnnotationSet>
    <sliceSequence>
      <slice
        location="PersonAnnotations_V1.xml"
        begin="2014-01-15" />
    </sliceSequence>
  </ontologyAnnotationSet>
</temporalOntologySchema>
```

Figure 7. The temporal schema on January 15, 2014.

```
<?xml version="1.0" encoding="UTF-8"?>
  <td:temporalRoot
    temporalSchemaLocation="PersonTemporalSchema.xml
" />
    <td:sliceSequence>
      <td:slice location="Persons_V1.rdf"
                begin="2014-01-15" />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 8. The temporal document on January 15, 2014.

```
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <nick_RepItem>
    <nick_Version>
      <timestamp_ValidExtent
               begin="2014-01-15" end="now" />
      <foaf:nick>Nor</foaf:nick>
    </nick_Version>
  </nick_RepItem>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <accountName_RepItem>
        <accountName_Version>
          <timestamp_ValidExtent
                 begin="2014-01-15" end="now" />
          <foaf:accountName>Nor_Tunsi
          </foaf:accountName>
        </accountName_Version>
      </accountName_RepItem>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
```

Figure 9. The squashed document correponding to the temporal document on January 15, 2014.

On February 08, 2014, the KBA updates the conventional ontology document "Persons_V1.rdf" as presented in Section II.A to produce a new conventional ontology document named "Persons_V2.rdf" (as in Figure 2). Since the conventional ontology schema (i.e., PersonSchema_V1.owl) and the ontology annotation document (i.e., PersonAnnotations_V1.xml) are not changed, the temporal ontology schema (i.e., PersonTemporalSchema.xml) is consequently not updated. However, the system updates the temporal document, in order to include the new slice of the new conventional ontology document, as shown in Figure 10. The squashed version of the updated temporal document is provided in Figure 11.

```
<?xml version="1.0" encoding="UTF-8"?>
  <td:temporalRoot
    temporalSchemaLocation="PersonTemporalSchema.xml
" />
    <td:sliceSequence>
      <td:slice location="Persons_V1.rdf"
                begin="2014-01-15" />
      <td:slice location="Persons_V2.rdf"
                begin="2014-02-08" />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 10. The temporal document on February 08, 2014.

```
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <nick_RepItem>
    <nick_Version>
      <timestamp_ValidExtent begin="2014-01-15"
                             end="2014-02-07" />
      <foaf:nick>Nor</foaf:nick>
    </nick_Version>
    <nick_Version>
      <timestamp_ValidExtent begin="2014-02-08"
                             end="now" />
      <foaf:nick>Nouri</foaf:nick>
    </nick_Version>
  </nick_RepItem>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <accountName_RepItem>
        <accountName_Version>
          <timestamp_ValidExtent
                    begin="2014-01-15"
                    end="2014-02-07"/>
          <foaf:accountName>Nor_Tunsi
          </foaf:accountName>
```

```
      </accountName_Version>
      <accountName_Version>
        <timestamp_ValidExtent
                   begin="2014-02-08"
                   end="now" />
        <foaf:accountName>Nouri_Tunsi
        </foaf:accountName>
      </accountName_Version>
    </accountName_RepItem>
  </foaf:OnlineAccount>
 </foaf:holdsAccount>
</foaf:Person>
```

Figure 11. The squashed document correponding to the temporal document on February 08, 2014.

Obviously, each one of the squashed documents (see Figure 9 and Figure 11) should conform to a particular schema, which is the representational schema (box 13), which is generated (by the Representational Schema Generator) from the temporal schema shown in Figure 7.

The example will be completed in Section V.D, after that the management of schema changes has been introduced.

IV. Our Approach to Schema Versioning in the τOWL Framework

In this section, we describe how τOWL conventional schema and τOWL logical and physical annotations can be versioned in our approach.

The first step of a schema versioning sequence is the creation of a first schema version: the KBA creates a conventional ontology schema (i.e., an OWL 2 file) and annotates it with some logical and physical annotations in an independent document (which is stored as an XML file), through, for instance, a graphical interface. Consequently, the system creates the temporal ontology schema (also stored as an XML file) that ties together the conventional schema and the annotations.

In further steps of the versioning sequence, applied when necessary, the KBA can independently change the conventional ontology schema, the logical or the physical ontology annotations.

Changing the conventional ontology schema leads to a new version of it. Similarly, changing logical or physical ontology annotations leads to a new version of the whole ontology annotation document. Therefore, the temporal ontology schema is implicitly and automatically updated by the system after each change of the conventional schema or of the annotation document.

Schema change operations performed by the KBA are high-level, since they are usually conceived having in mind high-level real-world specifications. Each of these high-level schema change operations is then mapped onto a sequence of low-level schema change operations (or schema change primitives). The mapping is performed by a schema change processor.

Each high-level change can be expressed as a sequence of change primitives. Thus, the consistency of the resulting conventional ontology schema (respectively, the resulting ontology annotation document or the resulting temporal ontology schema) is always guaranteed, if change primitives preserve the conventional ontology schema (respectively, the ontology annotation document or the temporal ontology schema) consistency.

Notice that in our approach, like in [15], the temporal schema, which ties the conventional schema and the annotations together, is not "explicitly" versioned; for each conventional schema (i.e., all the versions of this schema) and its associated annotation document (i.e., all the versions of this document), there is always one XML document that represents the temporal schema, which is updated when the conventional schema and/or the annotation document are changed. In fact, in the τOWL framework, the temporal schema is instrumental to support versioning of anything can change in the managed Semantic Web repository. As a consequence, by its nature, the temporal schema comes out "implicitly" versioned (i.e., all versions of a temporal schema document are stored within this document; the version of a temporal schema, valid at any given time Tx, could be extracted from that schema by removing all the <slice ... begin=Ty/> elements where Ty>Tx). Thus, we think that other kinds of versioning of the temporal schema are neither necessary nor could be meaningfully put at user's disposal (without getting out of the τOWL framework).

Notice also that neither conventional schema versioning nor annotation versioning lead automatically to proliferation of schema versions. The creation of a new conventional schema version (or of a new annotation document version) is anyway a seldom task during the Semantic Web repository lifetime, which can only be performed by an administrator of this repository. This task may consist of dozens of schema change operations, which are grouped together in the same single transaction.

V. Primitives for Changing Conventional Schema and Temporal Schema in the τOWL Framework

In this section, we first present our design principles and then introduce our proposed change primitives. We start by providing change primitives acting on conventional schema in τOWL and then we provide primitives for changing the temporal ontology schema. We have individuated change primitives (i.e., non-further decomposable in terms of the other ones), which make up a complete set of changes (i.e., such that any possible complex change can be defined via a combination/sequence of them). For each change primitive, we describe its arguments and its operational semantics. Finally, we give an example that illustrates the use of these primitives for versioning of τOWL conventional schema.

A. Design principles

The definition of the primitives will obey the following principles and conventions:

1) all primitives must work on a well-formed and valid Conventional Ontology Schema (COS) (or on the Temporal Ontology Schema (TOS)), that is, primitives must have a well-formed and valid COS (or TOS) as input and produce a well-formed and valid COS (or TOS) as output;

2) all primitives need to work on an OWL 2 file (or an XML file) storing the COS (or TOS), whose name must be supplied as argument;

3) for all primitives, arguments that are used to identify the object on which the primitive works are in the first place of the argument list;

4) primitives adding elements with possibly optional attributes have the values for all the attributes as arguments; empty places in the argument list stand for unspecified optional attributes;

5) for primitives changing elements, values are specified only for attributes that are changed; the value "unchanged" means that the corresponding attribute is not updated; an empty place in the argument list means that the corresponding attribute receives a nil value.

The lists of operations in the subsections that follow are the applications of the design principles presented above.

B. Primitives for changing conventional schema

Based on the OWL 2 ontology definition we adopted in Section III (e.g., assuming the signature O = {E, A, Exp}), we define a complete set of primitives for changing a conventional ontology schema, composed of twenty-eight operations. The idea is that each primitive deals with an OWL 2 ontology component (e.g., a class, a data property, an object property), by creating, removing or modifying such a component. For each primitive change, we describe its arguments and its operational semantics. Obviously, each primitive change has an effect on the COS. We do not present in this paper the effects of all primitive changes. We give only the effect of some selected primitive changes.

We have organized the proposed primitives into eight categories: (i) primitives acting on the whole COS (in the sub-section V.B.1), (ii) primitives acting on a class (in the sub-section V.B.2), (iii) primitives acting on a data property (in the sub-section V.B.3), (iv) primitives acting on an object property (in the sub-section V.B.4), (v) primitives acting on an annotation property (in the sub-section V.B.5), (vi) primitives acting on an entity axiom (in the sub-section V.B.6), (vii) primitives acting on a key axiom (in the sub-section V.B.7), and (viii) primitives acting on an entity expression (in the sub-section V.B.8).

1) Primitives acting on the whole COS

We have only three primitives:

- **CreateConventionalOntologySchema(COS.owl)**

It produces a valid empty OWL 2 file. According to the second design principle, the argument is the name of the OWL 2 file where the new COS is stored.

Notice also that the name of this file is the name of the ontology (e.g., Author, Paper, and Conference).

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="" />
</rdf:RDF>
```

- **RenameConventionalOntologySchema(oldCOS, newCOS)**

It changes the name of a COS from "oldCOS" to "newCOS" (or, it changes the name of an ontology from "oldOntoName" to "newOntoName").

- **DropConventionalOntologySchema(COS.owl)**

It removes the COS.owl file from disk, with the constraint that the argument represents an empty COS (i.e., like the one above initially created by the CreateConventionalOntologySchema primitive). Any other contents must have been removed before.

2) Primitives acting on a class

We have defined three primitives:

- **AddClass(COS.owl, className)**

It adds a new class having the name "className" to the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="className"/>
  </owl:Ontology>
</rdf:RDF>
```

- **RenameClass(COS.owl, oldClassName, newClassName)**

It changes the name of a class from "oldClassName" to "newClassName", in the COS.

- **DropClass(COS.owl, className)**

It removes the class having the name "className" from the COS.

3) Primitives acting on a data property

We have defined five primitives:

- **AddDataProperty(COS.owl, className, DataPropertyName, DataPropertyType)**

It adds a new data property having the name "DataPropertyName" and the type "DataPropertyType" to the class "className", in the COS.

Notice that the "className" and the "DataPropertyType" are considered as the "DataPropertyDomain" and the "DataPropertyRange", respectively.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="className"/>
    <owl:DatatypeProperty
rdf:about="DataPropertyName">
      <rdfs:domain rdf:resource="className"/>
      <rdfs:range rdf:resource="DataPropertyType"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

- **DropDataProperty(COS.owl, className, DataPropertyName)**

It removes the data property having the name "DataPropertyName" from the class "className", in the COS.

- **RenameDataProperty(COS.owl, className, oldDataPropertyName, newDataPropertyName)**

It changes the name of a data property from "oldDataPropertyName" to "newDataPropertyName" in the

class “className”, in the COS.

- **ChangeDataPropertyDomain(COS.owl, className, DataPropertyName, newDataPropertyDomain)**

It replaces the domain (or class) “className” of the data property “DataPropertyName” with a new domain “newDataPropertyDomain”, in the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="newDataPropertyDomain"/>
    <owl:DatatypeProperty
rdf:about="DataPropertyName">
      <rdfs:domain
rdf:resource="newDataPropertyDomain"/>
      <rdfs:range rdf:resource="DataPropertyType"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

- **ChangeDataPropertyRange(COS.owl, className, DataPropertyName, oldDataPropertyRange, newDataPropertyRange)**

It replaces the range (or type) “oldDataPropertyRange” of the data property “DataPropertyName” of the class “className” with a new range “newDataPropertyRange”, in the COS.

4) Primitives acting on an object property

We have defined five primitives:

- **AddObjectProperty(COS.owl, ObjectPropertyName, ObjectPropertyDomain, ObjectPropertyRange)**

It creates an object property (a relation) having the name “ObjectPropertyName” between a source class “ObjectPropertyDomain” and a target class “ObjectPropertyRange”, in the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    …
    <owl:ObjectProperty
rdf:about="ObjectPropertyName">
      <rdfs:domain
rdf:resource="ObjectPropertyDomain"/>
      <rdfs:range
rdf:resource="ObjectPropertyRange"/>
    </owl:ObjectProperty>
  </owl:Ontology>
</rdf:RDF>
```

- **DropObjectProperty(COS.owl, ObjectPropertyName)**

It removes the object property “ObjectPropertyName” from the COS.

- **RenameObjectProperty(COS.owl, oldObjectPropertyName, newObjectPropertyName)**

It changes the name of an object property from “oldObjectPropertyName” to “newObjectPropertyName”, in the COS.

- **ChangeObjectPropertyDomain(COS.owl, ObjectPropertyName, oldObjectPropertyDomain, newObjectPropertyDomain)**

It replaces the domain “oldObjectPropertyDomain” of the object property “ObjectPropertyName” with a new domain “newObjectPropertyDomain”, in the COS.

- **ChangeObjectPropertyRange(COS.owl, ObjectPropertyName, oldObjectPropertyRange, newObjectPropertyRange)**

It replaces the range “oldObjectPropertyRange” of the object property “ObjectPropertyName” with a new range “newObjectPropertyRange”, in the COS.

5) Primitives acting on an annotation property

We have defined three primitives:

- **AddAnnotationProperty(COS.owl, propertyType, propertyName, annotationProperty)**

It defines a new annotation property “annotationProperty” on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS.

- **DropAnnotationProperty(COS.owl, propertyType, propertyName, annotationProperty)**

It removes the annotation property “annotationProperty” defined on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS.

- **ChangeAnnotationProperty(COS.owl, propertyType, propertyName, oldAnnotationProperty, newAnnotationProperty)**

It replaces the annotation property “oldAnnotationProperty” defined on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS, with a new annotation property “newAnnotationProperty”.

6) Primitives acting on an entity axiom

We have defined three primitives:

- **AddEntityAxiom(COS.owl, entityType, entityName, entityAxiom)**

It defines a new entity axiom “entityAxiom” on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS.

- **DropEntityAxiom(COS.owl, entityType, entityName, entityAxiom)**

It removes the entity axiom “entityAxiom” defined on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS.

- **ChangeEntityAxiom(COS.owl, entityType, entityName, oldEntityAxiom, newEntityAxiom)**

It replaces the entity axiom “oldEntityAxiom” defined on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS, with a new entity axiom “newEntityAxiom”.

7) Primitives acting on a key axiom

We have defined also three primitives:

- **AddKeyAxiom(COS.owl, className, keyAxiom)**

It defines a new key axiom “keyAxiom” on the class “className”, in the COS.

- **DropKeyAxiom(COS.owl, className, keyAxiom)**

It removes the key axiom "keyAxiom" defined on the class "className", in the COS.

- **ChangeKeyAxiom(COS.owl, className, oldKeyAxiom, newKeyAxiom)**

It replaces the key axiom "oldKeyAxiom" defined on the class "className" in the COS, with a new key axiom "newKeyAxiom".

8) Primitives acting on an entity expression

We have only three primitives:

- **AddEntityExpression(COS.owl, entityType, entityName, entityExpression)**

It defines a new entity expression "entityExpression" on the entityType (i.e., Class, DataProperty, or ObjectProperty) named "entityName", in the COS.

- **DropEntityExpression(COS.owl, entityType, entityName, entityExpression)**

It removes the entity expression "entityExpression" defined on the entityType (i.e., Class, DataProperty, or ObjectProperty) named "entityName", in the COS.

- **ChangeEntityExpression(COS.owl, entityType, entityName, oldEntityExpression, newEntityExpression)**

It replaces the entity expression "oldEntityExpression" defined on the entityType (i.e., Class, DataProperty, or ObjectProperty) named "entityName", in the COS, with a new entity expression "newEntityExpression".

C. Primitives for changing the temporal ontology schema

Changing the temporal ontology schema is a task that must be done within the same transaction that changes the corresponding conventional ontology schema and/or the ontology annotation document. We also propose in this sub-section a complete set of primitives acting on a temporal ontology schema (their total number is four). For each primitive, we provide specifications for its actions and explanation of its parameters. We also present the effects of some of them. These primitives are as follows:

- **CreateTemporalOntologySchema(TOS.xml)**

It produces a valid empty TOS. According to the second design principle, the argument is the name of the XML file where the new TOS is stored.

The effect of the CreateTemporalOntologySchema(TOS.xml) primitive, that is, the contents of the COS.xml file after its application, is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema/>
```

- **DropTemporalOntologySchema(TOS.xml)**

It removes the TOS.xml file from disk, with the constraint that the argument represents an empty TOS (i.e., like the one above initially created by the CreateTemporalOntologySchema primitive). Any other contents must have been removed before.

- **AddSlice(TOS.xml, toWat, sourceSlice, targetSlice)**

It adds the <slice/> element with specified sourceSlice and targetSlice to the toWhat (i.e., <conventionalOntologySchema/> or <ontologyAnnotationSet/>) container.

- The sourceSlice parameter could be:

a) The keyword empty; in this case, the resource pointed by targetSlice is initialized to an empty conventional ontology schema or ontology annotation document according to the toWhat value.

b) The keyword current; in this case, the resource pointed by targetSlice is initialized with a copy of the current conventionalOntologySchema or ontologyAnnotationSet resource (according to toWhat), whose location is found in the TOS.xml temporal schema file by choosing the slice with the maximum value of begin in the corresponding sliceSequence (note: after the creation of the first schema version, this is the normal case).

c) A specified file name (URL): in this case, a copy of the specified resource is renamed as targetSlice and used as the new location (e.g., this case is used to create a new conventional ontology schema version from an already existing OWL 2 file, which could be quite common when creating the first schema version but can be used also later for reuse purpose and/or integrating independently developed schemata into a τOWL framework).

- The targetSlice parameter is the value assigned to the location attribute of <slice/> and must not correspond to the URL of any already existing OWL 2 file/resource.

For example, the effects of the AddSlice("TOS.xml", conventionalOntologySchema, empty, "COS_V1.owl") primitive are described in the following:

i) The contents of the TS.xml file is updated as follows (the transaction time associated to the execution of the transaction that includes this primitive is March 01, 2012, which is used as value of begin in the <slice/> element):

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location="COS_V1.owl"
             begin="2012-03-01" />
    </sliceSequence>
  </conventionalOntologySchema>
</temporalOntologySchema>
```

ii) A new empty conventional ontology schema, titled "COS_V1.owl", is created as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="" />
</rdf:RDF>
```

- **DropSlice(TOS.xml, fromWat, targetSlice)**

It removes the <slice/> element with specified targetSlice from the fromWhat (i.e., <conventionalOntologySchema/> or <ontologyAnnotationSet/>) container.

D. Running example conclusion

Let us resume the example started in Section II.A. Suppose that on July 18, 2014, the KBA decides to make some changes to the first version of the conventional ontology schema, in order to meet some changes in the code

of the application that exploit such an ontology schema. These changes are as follows:

– define an irreflexive relationship (or object property), named “childOf”, on the class “Person”;

– create two new classes, named “Man” and “Woman”, which inherit from the class “Person”;

– define a symmetric relationship, named “hasSpouse”, between the class “Man” and the class “Woman”;

– specify a relationship, named “hasWife”, between the class “Man” and the class “Woman”. This relationship inherits from the relationship “hasSpouse”;

– change the name of the property (or data property) “name” of the class “Person” to “fullName”;

– add a new property, named “age” and having the XSD type “nonNegativeInteger”, to the class “Person”;

– specify an expression on the relationship “holdsAccount”, which indicates that each person must have at least one online account.

The second version of the conventional ontology schema and the second version of each one the two conventional ontology instance documents are shown in Figure 12, Figure 13, and Figure 14, respectively. The temporal ontology schema is also updated by adding a new slice related to this new version of the conventional ontology schema, as shown in Figure 15. Moreover, the temporal document is updated, in order to include two new slices corresponding to the two new conventional ontology instance documents, as shown in Figure 16. The squashed version of the updated temporal document that consequently can be generated by the Temporal Instances Generator tool is similar to documents provided in Figure 9 and Figure 11. Notice that changes are presented in red, in Figures 12-16.

```
<rdf:RDF>
  <owl:Ontology
rdf:about="http://purl.org/az/foaf#">
   <owl:Class rdf:about="Person"/>
   <owl:Class rdf:about="OnlineAccount"/>
   <owl:Class rdf:about="Man"/>
   <owl:Class rdf:about="Woman"/>
   <owl:DatatypeProperty rdf:about="accountName">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:about="nick">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:about="fullName">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:about="age">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#nonNegativeI
nteger"/>
   </owl:DatatypeProperty>
   <owl:ObjectProperty rdf:about="holdsAccount">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource="OnlineAccount"/>
   </owl:ObjectProperty>
   <owl:ObjectProperty rdf:about="hasSpouse">
     <rdfs:domain rdf:resource="Man"/>
     <rdfs:range rdf:resource="Woman"/>
   </owl:ObjectProperty>
   <owl:ObjectProperty rdf:about="hasWife">
     <rdfs:domain rdf:resource="Man"/>
     <rdfs:range rdf:resource="Woman"/>
   </owl:ObjectProperty>
   <owl:ObjectProperty rdf:about="childOf">
     <rdfs:domain rdf:resource="Person"/>
     <rdfs:range rdf:resource="Person"/>
   </owl:ObjectProperty>
   <owl:Class rdf:about="Man">
     <rdfs:subClassOf rdf:resource="Person"/>
   </owl:Class>
   <owl:Class rdf:about="Woman">
     <rdfs:subClassOf rdf:resource="Person"/>
   </owl:Class>
   <owl:SymmetricProperty rdf:about="hasSpouse"/>
   <owl:IrreflexiveProperty rdf:about="childOf"/>
   <owl:ObjectProperty rdf:about="hasWife">
     <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
   </owl:ObjectProperty>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#holdsAccount"/>
     <owl:minCardinality rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#nonNegativeI
nteger">1
     </owl:minCardinality>
   </owl:Restriction>
   …
  </owl:Ontology>
</rdf:RDF>
```

Figure 12. Second version of the conventional ontology schema (PersonSchema_V2.owl), on July 18, 2014.

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:fullName>Nouredine Tounsi</foaf:fullName>
  <foaf:nick>Nor</foaf:nick>
  <age/>
  <childOf/>
  <hasSpouse/>
  <hasWife/>
  <foaf:holdsAccount>
    <foaf:OnlineAccount rdf:about=
    "https://www.facebook.com/Nouredine.Tounsi">
      <foaf:accountName>Nor_Tunsi</foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 13. “Persons_V3.rdf”: the second version of the conventional ontology instance document “Persons_V1.rdf”, on July 18, 2014.

```
…
<foaf:Man rdf:ID="#Person1">
  <foaf:fullName>Nouredine Tounsi</foaf:fullName>
  <foaf:nick>Nouri</foaf:nick>
  <age/>
  <childOf/>
  <hasSpouse/>
  <hasWife/>
```

```
  <foaf:holdsAccount>
    <foaf:OnlineAccount rdf:about=
       "https://www.facebook.com/Nouredine.Tounsi">
      <foaf:accountName>Nouri_Tunsi
      </foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 14. “Persons_V4.rdf”: the second version of the conventional ontology instance document “Persons_V2.rdf”, on July 18, 2014.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location=”PersonSchema_V1.owl”
begin=”2014-01-15” />
      <slice location=”PersonSchema_V2.owl”
begin=”2014-07-18” />
    </sliceSequence>
  </conventionalOntologySchema>
  <ontologyAnnotationSet>
    <sliceSequence>
      <slice location=”PersonAnnotations_V1.xml”
begin=”2014-01-15” />
    </sliceSequence>
  </ontologyAnnotationSet>
</temporalOntologySchema>
```

Figure 15. The temporal ontology schema (PersonTemporalSchema.xml), on July 18, 2014.

```
<?xml version=”1.0” encoding=”UTF-8”?>
  <td:temporalRoot
    temporalSchemaLocation=”PersonTemporalSchema.xml
” />
    <td:sliceSequence>
      <td:slice location=”Persons_V1.rdf”
                begin=”2014-01-15” />
      <td:slice location=”Persons_V2.rdf”
                begin=”2014-02-08” />
      <td:slice location=”Persons_V3.rdf”
                begin=”2014-07-18” />
      <td:slice location=”Persons_V4.rdf”
                begin=”2014-07-18” />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 16. The temporal document (PersonTemporalDocument.xml) on July 18, 2014.

The transaction listed in the following contains the sequence of primitives that have been performed on the temporal ontology schema (PersonTemporalSchema.xml, in Figure 7), on the first version of the conventional ontology schema (PersonSchema_V1.owl in Figure 4), on the first version of the conventional ontology instance document (Persons_V1.rdf in Figure 1) and on the second version of the conventional ontology instance document (Persons_V2.rdf in Figure 2), in order to update the temporal ontology schema (see Figure 15) and the temporal document (see Figure 16) and to produce the second version of the conventional ontology schema, named “PersonSchema_V2.owl” (see Figure 12), and the third and the fourth versions of the conventional ontology instance document, named “Persons_V3.rdf” (see Figure 13) and “Persons_V4.rdf” (see Figure 14), respectively, which are valid with respect to “PersonSchema_V2.owl”:

```
Begin Transaction
(i) AddSlice(“PersonTemporalSchema.xml”,
   conventionalOntologySchema, current,
   “PersonSchema_V2.owl”)
(ii) AddObjectProperty(“PersonSchema_V2.owl”,
   “childOf”, “Person”, “Person”)
(iii) AddEntityAxiom(“PersonSchema_V2.owl”,
   ObjectProperty, “childOf”,
   “IrreflexiveProperty”)
(iv) AddClass(“PersonSchema_V2.owl”, “Man”)
(v) AddClass(“PersonSchema_V2.owl”, “Woman”)
(vi) AddEntityAxiom(“PersonSchema_V2.owl”, Class,
   “Man”, “subClassOf(Person)”)
(vii) AddEntityAxiom(“PersonSchema_V2.owl”, Class,
   “Woman”, “subClassOf(“Person”)”)
(viii) AddObjectProperty(“PersonSchema_V2.owl”,
   “hasSpouse”, “Man”, “Woman”)
(ix) AddEntityAxiom(“PersonSchema_V2.owl”,
   ObjectProperty, “hasSpouse”,
   “SymmetricProperty”)
(x) AddObjectProperty(“PersonSchema_V2.owl”,
   “hasWife”, “Man”, “Woman”)
(xi) AddEntityAxiom(“PersonSchema_V2.owl”,
   ObjectProperty, “hasWife”,
   “subObjectPropertyOf(“hasSpouse”)”)
(xii) RenameDataProperty(“PersonSchema_V2.owl”,
   “Person”, “name”, “fullName”)
(xiii) AddDataProperty(“PersonSchema_V2.owl”,
   “Person”, “age”,
   “http://www.w3.org/2001/XMLSchema#nonNegativeIn
   teger”)
(xiv) AddEntityExpression(“PersonSchema_V2.owl”,
   ObjectProperty, “holdsAccount”,
   “minCardinality(1)”)
Commit
```

The transaction time associated to the execution of the transaction above is July 18, 2014, which is used as value of the attribute “begin” of the new <slice/> element, corresponding to the new conventional ontology schema version, in the temporal ontology schema file.

Notice that on July 18, 2014, our multiversion τOWL framework is thus composed of two successive versions of the conventional ontology schema (shown in Figure 4 and Figure 12, respectively), four versions of the conventional ontology instance documents (shown in Figure 1, Figure 2, Figure 13, and Figure 14, respectively), one version of the ontology annotation document (shown in Figure 6), the temporal document (shown in Figure 16) and the temporal ontology schema (shown in Figure 15).

Notice also that “Persons_V3.rdf” (shown in Figure 13) and “Persons_V4.rdf” (shown in Figure 14) are the results of schema change propagation (i.e., the effects of schema changes on instances), in order to adapt all existing instances, stored in “Persons_V1.rdf” (shown in Figure 1) and “Persons_V2.rdf” (shown in Figure 2), to the new schema version “PersonSchema_V2.owl” (shown in Figure 12). In fact, after creating "Persons_V3.rdf" as a copy of "Persons_V1.rdf" and "Persons_V4.rdf" as a copy of "Persons_V2.rdf", the two following XQuery Update Facility [31] statements could be executed on "Persons_V3.rdf" and "Persons_V4.rdf", respectively, to achieve the purpose:

```
for $p in fn:doc("Persons_V3.rdf")//foaf:Person
return {
   rename node $p/foaf:name as "foaf:fullName",
   insert node <age/> after $p/foaf:nick,
   insert node <childOf/> after $p/age,
   insert node <hasSpouse/> after $p/childOf,
   insert node <hasWife/> after $p/hasSpouse
}
for $p in fn:doc("Persons_V4.rdf")//foaf:Person
return {
   rename node $p/foaf:name as "foaf:fullName",
   insert node <age/> after $p/foaf:nick,
   insert node <childOf/> after $p/age,
   insert node <hasSpouse/> after $p/childOf,
   insert node <hasWife/> after $p/hasSpouse
}
```

These statements are derived, in an automatic and transparent way, by the system as a part of the semantics of the schema change primitives. They are not part of what the KBA puts in his/her schema change transaction, but it is the system to generate and add them to the transaction that is actually executed.

VI. Related Work Discussion

In the literature, there are several proposals that deal with managing temporal aspects in ontologies or Semantic Web. OWL-Time (formerly DAML-Time) [32] is a temporal ontology that has been developed for describing the temporal content of Web pages and the temporal properties of Web services. Excepting language constructs for representing time in ontologies, mechanisms for representing evolution of concepts (e.g., events) over time are absent. Furthermore, temporal relations cannot be expressed directly in OWL, since they are ternary (i.e., properties of objects that change in time involve also a temporal value in addition to the object and the subject); representing such temporal relations in OWL requires appropriate methods (e.g., 4D-fluents [33]). Our approach allows a KBA to represent (i) evolution of concepts over time, and (ii) temporal relations.

In [34], the authors present the annotation features of OWL 2 by showing that it allows for annotations on ontologies, entities, anonymous individuals, axioms (e.g., giving information about who asserted an axiom or when), and annotations themselves. In our work, we took another direction from using OWL 2 annotation features because we rather wanted to exploit the power of the τXSchema approach (e.g., including the exploitation of a τXSchema-like underlying infrastructure).

Time dimension(s) are explicitly added to Semantic Web languages and formalisms (e.g., RDF, OWL and SPARQL [35]) in order to represent time in semantic annotations, to build temporal ontologies and to support temporal querying and reasoning. An annotated bibliography of previous work in this area is presented in [13], and a survey on the models and query languages for temporally annotated RDF is provided in [36]. In particular, in the literature, there are various contributions that propose to represent temporal data in the Semantic Web.

Gutiérrez et al. [37] presented a comprehensive framework to incorporate temporal reasoning into RDF, yielding temporal RDF graphs. They define a syntactic notion of temporal RDF graphs. A powerful system, called CHRONOS, for reasoning over temporal information in OWL ontologies is presented in [38]. Since qualitative representations are very common in natural language expressions such as in free text or speech and can be proven to be valuable in the Semantic Web, the authors choose to represent both qualitative temporal (i.e., information whose temporal extents are unknown such as “before”, “after” for temporal relations) and quantitative information (i.e., where temporal information is defined precisely, e.g., using dates). The CHRONOS reasoner can be applied to temporal relations in order to infer implied relations and to detect inconsistencies while retaining soundness, completeness and tractability over the supported relations set. The paper [39] proposes a logic-based approach to introduce valid-time into RDFS and OWL 2 languages. An extension of SPARQL that can be used to query temporal RDF(S) and OWL 2 is also presented. Moreover, the author describes a general query evaluation algorithm that can be used with all entailment relations used in the Semantic Web. Finally, he presents two optimizations of the algorithm that are applicable to entailment relations characterized by a set of deterministic rules, such RDF(S) and OWL 2 RL/RDF Entailment. As opposed to Gutiérrez et al. [37], Anagnostopoulos et al. [38] and Motik [39], in our present approach, we are not interested in temporal (or spatio-temporal) reasoning.

Two complementary and alternative proposals for modeling temporally changing information in OWL are proposed in [40]. They are based on the perdurantist theory and benefit from results coming from the discipline of Formal Ontology, in order to restrict the appropriate use of the proposed frameworks. In the first proposal, the authors combine the perdurantist worm view with the notion of individual concepts for formulating a conceptual structure that allows one to separate, from the information that define all the individuals, the information concerning those that can possibly change. In the second proposal, they extend the first proposal with the distinction between objects and moments and the notion of qua individuals, where a qua individual is the way an object participates in a certain relation.

Differently from Zamborlini et al. [40], our approach does not deal with modeling of time inside the ontology, but just aims at supporting temporal versioning.

O'Connor et al. [41] present a methodology and a set of tools for representing and querying temporal information in OWL ontologies. Their approach uses a lightweight temporal model to encode the temporal dimension of data. It also uses the OWL-based Semantic Web Rule Language (SWRL) and the SWRL-based OWL query language (SQWRL) to reason with and query the temporal information represented using the proposed model. By now, our approach does not support temporally-aware semantic rules.

The authors of [42] propose a new language, called temporal OWL (tOWL), which is an extension of the Ontology Web Language Description Logics (OWL-DL) to the temporal aspect. It enables the representation of time and change in dynamic domains. Through a layered approach, they introduce three extensions: (i) Concrete Domains, which allow the representation of restrictions using concrete domain binary predicates, (ii) Temporal Representation, which introduces timepoints, relations between timepoints, intervals, and Allen's 13 interval relations [43] into the language, and (iii) TimeSlices/Fluents, which implement a perdurantist view on individuals and enable the representation of complex temporal aspects such as process state transitions. The main purpose of our approach is to support past ontology versions, to be accessed via time-slice queries. We think that supporting temporal ontology versions is very interesting for several purposes and in different areas. The problem of not having temporal versions is that, e.g., if we have now to investigate on someone having put some illegal material on Facebook last week, we want to be able to individuate the account details even if they have been changed thereafter.

As far as ontology schema evolution and versioning problems are concerned, we can find also several studies which have dealt with them. In general, we could summarize them under the three following groups of issues taken into account:

– modeling, implementing, and detecting changes in ontologies [44][45][46][47][48];

– preserving the consistency of evolving ontologies [49][50][51][52];

– ontology versioning support [53][54][55][56][57][58][59][60][61].

Our approach belongs to the last set of contributions. In [53], the authors consider the notion of context as an abstraction mechanism to deal with multi-representation ontologies (contextual ontologies). A formal representation language based on modal description logics is proposed to comply with the requirements of multiple perspectives of domain ontology.

Bouquet et al. [54] show how ontologies can be contextualized, by proposing Context OWL (C-OWL), a language whose syntax and semantics have been obtained by extending the OWL syntax and semantics to allow for the representation of contextual ontologies. Notice that an ontology is said to be contextualized when its contents are kept local, and, therefore, not shared with other ontologies, and mapped with the contents of other ontologies via explicit (context) mappings.

Heflin et al. [55] show that the Semantic Web needs a formal semantics for the various kinds of links between ontologies and other documents, and then provide a model theoretic semantics that takes into account ontology extension and ontology versioning.

Völkel et al. [56] present an RDF-centric versioning approach and an implementation called SemVersion. The proposed approach separates the management aspects from the versioning core functionality. SemVersion provides structural and semantic versioning for RDF models and RDF-based ontology languages like RDFS, considering blank node enrichment as a technique to identify the blank nodes in the versioned models.

Bedi et al. [57] introduce an approach that combines the concepts of temporal frame and slot versioning with the ontology to create temporal tagged ontologies with embedded versioning. The authors also propose to enhance the existing OWL to enable the creation of temporal tagged OWL ontologies: two new tags, "rdf:Validity" and "rdf:Timestamp", are introduced and a scheme is presented for the value of the "rdf:Id" and "rdf:Resource" tags to make the temporal tagged ontologies consistent with the non-temporal ontologies.

Kondylakis et al. [61] propose a solution that allows query answering in data integration systems under evolving ontologies without mapping redefinition. This is achieved by rewriting queries among ontology versions and then forwarding them to the underlying data integration systems to be answered.

The works that are more strictly related with our approach are [58], [59], and [60]. Grandi [58] provides a multi-temporal RDF database model; a database consists in a set of RDF triples timestamped along the valid and/or transaction time axes. The data model is equipped with manipulation operations which allow the KBA to maintain a multi-temporal RDF database in order to manage temporal versions of an ontology. Grandi et al. [59] introduce "The Valid Ontology", a framework to represent and store multiple temporal versions of an ontology in a compact temporal XML format and efficiently extract ontology snapshots from the multiversion XML document via a temporal XML processor. Grandi [60] focuses on temporal versioning of light-weight ontologies expressed in RDF(S) and show how the multi-temporal RDF data model proposed in [58] can be used to support RDF(S) ontology versioning. The data model is equipped with a complete set of primitive ontology change operations, which are defined in terms of low-level updates acting on RDF triples. When used within the transaction template, which has also been introduced, the proposed ontology changes allow a KBA to define and manage temporal versions of an RDF(S) ontology.

However, whereas all the works in this group, including [58], [59], and [60], basically propose *ad hoc* solutions for the management of temporal versions of RDF, RDF(S) or OWL resources, we introduce a τXSchema-like general framework embodying a disciplined and principled approach to temporal versioning of Semantic Web documents, both at

instance and at schema levels.

VII. Conclusion and Future Work

In this paper, we proposed τOWL, a τXSchema-like framework, which allows creating a temporal OWL 2 ontology from a conventional OWL 2 ontology and a set of logical and physical annotations. Our framework ensures logical and physical data independence, since it (i) separates conventional schema, logical annotations, and physical annotations, and (ii) allows each one of these three components to be changed independently and safely. Furthermore, adoption of τOWL provides for a low-impact solution, since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards. The extension of OWL 2 to temporal and versioning aspects is performed without having to depend on approval of proposed extensions by standardization committees (and on upgrade of existing tools conforming to standards to comply with approved extensions).

Moreover, we have extended our τOWL framework by proposing a general approach for schema versioning in it and focusing on the definition of a set of change primitives for supporting the evolution of both temporal and conventional ontology schema. Our approach helps the KBA in the management of conventional schema changes in τOWL-based Semantic Web repositories and guarantees the maintenance of a full history of evolving conventional ontology instances and schemata.

In order to embed our approach into a user-friendly environment at the disposal of KBAs, a tool for the management of temporal ontologies in the τOWL framework is under development at the University of Sfax. A first release of the tool, named τOWL-Manager [62], is already available and implements our τOWL framework with the support of temporal versioning of ontology instances. The new release currently under development will support all schema change primitives proposed in this paper, and put them at the disposal of KBAs, via an intuitive interface which assists them in expressing their needs to fulfill application requirements. Furthermore, we are also extending the present work by defining a complete set of schema change primitives for the ontology annotation document which stores logical and physical annotations specified on the conventional ontology schema.

Besides, in order to further simplify the work of KBAs and to make our approach more useful, we intend to propose in our future work high-level and more user-friendly schema change operations, based on the primitives introduced in this paper and on those that will be defined for changing annotations. A high-level operation is a valid sequence of primitives, which correspond to frequent schema evolution needs and allows expressing complex changes in a more compact way [63]. Moreover, we will also allow the KBA to build his/her own high-level schema change operations, by combining in a consistent way pre-defined high-level operations and/or primitives, through the use of a specific tool that will be integrated in a future release of the τOWL-Manager environment.

As a part of our future work, we will also thoroughly study the propagation of changes performed on conventional ontology schema, i.e., their effects on conventional ontology instances stored in conventional ontology instance documents, which are valid with respect to the conventional ontology schema.

Finally, we also plan to address querying of temporal ontology instances under schema versioning, in the τOWL framework. The starting point for this extension will be the T-SPARQL language [27], which allows end users and KBAs to express queries on multi-temporal ontology instances (which are composed of multi-temporal RDF triples) under a single ontology schema version; such a language could be extended with features to support schema versions and specify multi-schema queries, i.e., queries involving instances of several schema versions [64].

References

[1] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, "τOWL: A Framework for Managing Temporal Semantic Web Documents," Proceedings of the 8th International Conference on Advances in Semantic Processing (SEMAPRO 2014), Rome, Italy, 24-28 August 2014, pp. 33-41.

[2] C. S. Jensen and R. T. Snodgrass, "Temporal Data Management," IEEE Transactions on Knowledge and Data Engineering, vol. 11, January/February 1999, pp. 36-44.

[3] O. Etzion, S. Jajodia, and S. Sripada (eds.), "Temporal Databases: Research and Practice," LNCS 1399, Springer-Verlag, 1998.

[4] C. S. Jensen and R. T. Snodgrass, "Temporal Database," in Liu L., Özsu M.T., (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 2957-2960.

[5] F. Grandi, "Temporal Databases," in M. Koshrow-Pour, (Ed.), Encyclopedia of Information Science and Technology (3rd Ed.), IGI Global, Hershey, in press.

[6] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World Wide Web," Communications of the ACM, vol. 37, August 1994, pp. 76-82.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," Scientific American, vol. 284, May 2001, pp. 34-43.

[8] W3C Semantic Web Activity. <http://www.w3.org/2001/sw/> [retrieved: May, 2015]

[9] N. Guarino (Ed.), Formal Ontology in Information Systems, IOS Press, Amsterdam, 1998.

[10] W3C, OWL 2 Web Ontology Language – Primer (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-primer/> [retrieved: May, 2015]

[11] W3C, OWL 2 Web Ontology Language – Document Overview (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-overview/> [retrieved: May, 2015]

[12] W3C, OWL 2 Web Ontology Language – Profiles (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-profiles/> [retrieved: May, 2015]

[13] F. Grandi, "Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the Semantic Web," SIGMOD Record, vol. 41, December 2012, pp. 18-21.

[14] F. Currim, S. Currim, C. E. Dyreson, and R. T. Snodgrass, "A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with tXSchema," Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), Heraklion, Crete, Greece, 14-18

March 2004, pp. 348-365.

[15] R. T. Snodgrass, C. E. Dyreson, F. Currim, S. Currim, and S. Joshi, "Validating Quicksand: Schema Versioning in τXSchema," Data Knowledge and Engineering, vol. 65, May 2008, pp. 223-242.

[16] F. Currim, S. Currim, C. E. Dyreson, S. Joshi, R. T. Snodgrass, S. W. Thomas, and E. Roeder, "τXSchema: Support for Data- and Schema-Versioned XML Documents," TimeCenter Technical Report TR-91, 279 pages, September 2009. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-91.pdf> [retrieved: May, 2015]

[17] C. E. Dyreson and F. Grandi, "Temporal XML," in L. Liu and M. T. Özsu (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 3032-3035.

[18] Z. Brahmia, R. Bouaziz, F. Grandi, and B. Oliboni, "Schema Versioning in τXSchema-Based Multitemporal XML Repositories," Proceedings of the 5th IEEE International Conference on Research Challenges in Information Science (RCIS 2011), Guadeloupe - French West Indies, France, 19-21 May 2011, pp. 1-12.

[19] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Versioning of Conventional Schema in the τXSchema Framework," Proceedings of the 8th International Conference on Signal Image Technology & Internet Systems (SITIS'2012), Sorrento – Naples, Italy, 25-29 November 2012, pp. 510-518.

[20] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Schema Change Operations for Full Support of Schema Versioning in the τXSchema Framework," International Journal of Information Technology and Web Engineering, vol. 9, April-June 2014, pp. 20-46.

[21] T. Burns, E. Fong, D. Jefferson, R. Knox, L. Mark, C. Reedy, L. Reich, N. Roussopoulos, and W. Truszkowski, "Reference Model for DBMS Standardization, Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group," SIGMOD Record, vol. 15, March 1986, pp. 19-58.

[22] J. F. Roddick, "Schema Versioning," in Liu L., Özsu M.T., (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 2499-2502.

[23] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Schema Versioning," in M. Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology (3rd Ed.), IGI Global, 2014, pp. 7651-7661.

[24] D. Rogozan and G. Paquette, "Managing ontology changes on the semantic web," Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), Compiegne, France, 19-22 September 2005, pp. 430-433.

[25] The Friend of a Friend (FOAF) project. <http://www.foaf-project.org/> [retrieved: May, 2015]

[26] W3C, Resource Description Framework (RDF), Semantic Web Standard. <http://www.w3.org/RDF/> [retrieved: May, 2015]

[27] F. Grandi, "T-SPARQL: a TSQL2-like temporal query language for RDF," Proceedings of the 1st International Workshop on Querying Graph Structured Data (GraphQ 2010), Novi Sad, Serbia, 20 September 2010, pp. 21-30.

[28] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass, "On the Semantics of "Now" in Databases," ACM Transactions on Database Systems, vol. 22, June 1997, pp. 171–214.

[29] W3C, RDF/XML Syntax Specification (Revised), W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/> [retrieved: May, 2015]

[30] XML Schema Part 0: Primer Second Edition, W3C Recommendation, 28 October 2004. <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/> [retrieved: May, 2015]

[31] W3C, XQuery Update Facility 1.0, W3C Candidate Recommendation, 17 March 2011. <http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/> [retrieved: May, 2015]

[32] W3C, Time Ontology in OWL, W3C Working Draft, 27 september 2006. <http://www.w3.org/TR/owl-time/> [retrieved: May, 2015]

[33] C. A. Welty and R. Fikes, "A Reusable Ontology for Fluents in OWL," Proceedings of the 4th International Conference on Formal Ontology in Information Systems (FOIS 2006), Baltimore, Maryland, USA, 9-11 November 2006, pp. 226-236.

[34] W3C, OWL 2 Web Ontology Language – New Features and Rationale (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-new-features/> [retrieved: May, 2015]

[35] W3C, SPARQL Query Language for RDF, W3C Recommendation, 15 January 2008, <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/> [retrieved: May, 2015]

[36] A. Analyti and I. Pachoulakis, "A survey on models and query languages for temporally annotated RDF," International Journal of Advanced Computer Science and Applications, vol. 3, September 2012, pp. 28-35.

[37] C. Gutiérrez, C. A. Hurtado, and A. A. Vaisman, "Introducing time into RDF," IEEE Transactions on Knowledge and Data Engineering, vol. 19, February 2007, pp. 207-218.

[38] E. Anagnostopoulos, S. Batsakis, and E. G. M. Petrakis, "CHRONOS: A Reasoning Engine for Qualitative Temporal Information in OWL," Proceedings of the 17th International Conference in Knowledge-Based and Intelligent Information & Engineering Systems (KES 2013), Kitakyushu, Japan, 9-11 September 2013, pp. 70-77.

[39] B. Motik, "Representing and Querying Validity Time in RDF and OWL: A Logic-based Approach," Proceedings of the 9th International Semantic Web Conference (ISWC 2010), Shanghai, China, 7-11 November 2010, pp. 550-565.

[40] V. Zamborlini and G. Guizzardi, "On the representation of temporally changing information in OWL," Workshops Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOCW 2010), Vitória, Brazil, 25-29 October 2010, pp. 283-292.

[41] M. J. O'Connor and A. K. Das, "A method for representing and querying temporal information inOWL," In Biomedical Engineering Systems and Technologies, volume 127 of Communications in Computer and Information Science, pp. 97-110. Springer-Verlag, Heidelberg, Germany, 2011.

[42] V. Milea, F. Frasincar, and U. Kaymak, "tOWL: A Temporal Web Ontology Language," IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 42, February 2012, pp. 268-281.

[43] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," Communications of the ACM, vol. 26, November 1983, pp. 832-843.

[44] M. C. A. Klein and D. Fensel, "Ontology versioning on the Semantic Web," Proceedings of the 1st Semantic Web Working Symposium (SWWS 2001), Stanford University, California, USA, 30 July – 1 August 2001, pp. 75-91.

[45] M. C. A. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov, "Ontology Versioning and Change Detection on the Web," Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002), Siguenza, Spain, 1-4 October 2002, pp. 197-212.

[46] N. F. Noy and M. A. Musen, "Ontology versioning in an ontology management framework," IEEE Intelligent Systems, vol. 19, July 2004, pp. 6-13.

[47] J. Eder and C. Koncilia, "Modelling changes in ontologies," Proceedings of the OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, 25-29 October 2004, pp. 662-673.

[48] T. Redmond, M. Smith, N. Drummond, and T. Tudorache, "Managing change: an ontology version control system," Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008), Karlsruhe, Germany, 26-27 October 2008. CEUR Workshop Proceedings (CEUR-WS.org), Vol-432. <http://ceur-ws.org/Vol-432/owled2008eu_submission_33.pdf> [retrieved: May, 2015]

[49] P. De Leenheer, "Revising and managing multiple ontology versions in a possible worlds setting," Proceedings of the OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, 25-29 October 2004, pp. 798-809.

[50] P. Haase and L. Stojanovic, "Consistent Evolution of OWL Ontologies," Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Heraklion, Crete, Greece, 29 May – 1 June 2005, pp. 182-197.

[51] N. Sassi, W. Jaziri, and F. Gargouri, "How to Evolve Ontology and Maintain Its Coherence - A Corrective Operations-based Approach," Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD 2009), Funchal - Madeira, Portugal, 6-8 October 2009, pp. 384-387.

[52] W. Jaziri, N. Sassi, and F. Gargouri, "Approach and tool to evolve ontology and maintain its coherence," International Journal of Metadata, Semantics, and Ontologies, vol. 5, May 2010, pp. 151-166.

[53] A. Arara and D. Benslimane, "Towards formal ontologies requirements with multiple perspectives," Proceedings of the 6th International Conference on Flexible Query Answering Systems (FQAS 2004), Lyon, France, 24-26 June 2004, pp. 150-160.

[54] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt, "Contextualizing ontologies," Journal of Web Semantics, vol. 1, October 2004, pp. 325-343.

[55] J. Heflin and Z. Pan, "A model theoretic semantics for ontology versioning," Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan, 7-11 November 2004, pp. 62-76.

[56] M. Völkel and T. Groza, "SemVersion: An RDF-based Ontology Versioning System," Proceedings of the IADIS International Conference on WWW/Internet (ICWI 2006), Murcia, Spain, 5-8 October 2006, vol. 1, pp. 195-202. <http://www.xam.de/2006/10-SemVersion-ICIW2006.pdf> [retrieved: May, 2015]

[57] P. Bedi and S. Marwaha, "Versioning OWL ontology using temporal tags," Proceedings of the 21st International Conference on Computer, Electrical, Systems Science and Engineering (CESSE'07), Vienna, Austria, 25-27 May 2007, pp. 332-337.

[58] F. Grandi, "Multi-temporal RDF ontology versioning," Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD 2009), Washington DC, USA, 26 October 2009. CEUR Workshop Proceedings (CEUR-WS.org), Vol-519. <http://ceur-ws.org/Vol-519/grandi.pdf> [retrieved: May, 2015]

[59] F. Grandi and M. R. Scalas, "The valid ontology: A simple OWL temporal versioning framework," Proceedings of the 3rd International Conference on Advances in Semantic Processing (SEMAPRO 2009), Sliema, Malta, 11-16 October 2009, pp. 98-102.

[60] F. Grandi, "Light-weight Ontology Versioning with Multi-temporal RDF Schema," Proceedings of the 5th International Conference on Advances in Semantic Processing (SEMAPRO 2011), Lisbon, Portugal, 20-25 November 2011, pp. 42-48.

[61] H. Kondylakis and D. Plexousakis, "Ontology evolution without tears," Journal of Web Semantics, vol. 19, March 2013, pp. 42-58.

[62] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, "τOWL-Manager: A Tool for Managing Temporal Semantic Web Documents in the τOWL Framework," Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015), Nice, France, 19-24 July 2015, in press.

[63] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "High-level Operations for Changing Temporal Schema, Conventional Schema and Annotations, in the τXSchema Framework," TimeCenter Technical Report TR-96, 56 pages, January 2014. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-96.pdf> [retrieved: May, 2015]

[64] F. Grandi, "A relational multi-schema data model and query language for full support of schema versioning," Proceedings of SEBD 2002 – National Conference on Advanced Database Systems, Isola d'Elba, Italy, 19-21 June 2002, pp. 323-336.

Productivity-Based Software Estimation Models and Process Improvement: an Empirical Study

Alain Abran, Jean-Marc Desharnais
Department of Software Engineering & IT
École de technologie supérieure,
University of Québec,
Montreal, Canada
alain.abran@etsmtl.ca,
jean-marc.desharnais@etsmtl.ca

Mohammad Zarour
College of Computer and
Information Sciences
Prince Sultan University,
Riyadh, Saudi Arabia
mzarour@psu.edu.sa

Onur Demirörs
Middle East Technical University,
Ankara, Turkey
demirors@metu.edu.tr

***Abstract*—This paper proposes an approach to software estimation based on productivity models with fixed/variable costs and economies/diseconomies of scale. The paper looks first at productivity alone as a single variable model, and then discusses multi-variable models for estimation in specific contexts. An empirical study in a Canadian organization that illustrates the contribution of these concepts from economics in developing tailor-made estimation models based on the performance of the organization studied is presented, as well as the use of the SWEBOK Guide for the identification of process improvements areas.**

Keywords-Software economics; productivity models; fixed variable cost; estimation models; Function Point.

I. INTRODUCTION

Over the past 40 years researchers have approached software effort estimation using different mixes of cost drivers as well as various techniques that combine costs drivers with either expert opinion or mathematical models. The main goal is to produce 'accurate estimates', either intuitively based on expert opinion, or through mathematical models.

In contrast to traditional approaches and strategies in software engineering that focus strictly on estimation, this paper examines an approach common in economics that looks first at productivity alone as a single variable model, before moving on to multi-variable models for estimation in specific contexts. The paper expands on these concepts and reports on an empirical study that illustrates the contribution of these concepts from economics to develop tailor-made estimation models based on the performance of the organization studied. This paper reports in more detail on the empirical study presented briefly in [1], including data quality controls, functional size measurement and the identification of process improvement based on the SWEBOK Guide [2]. Some of the concepts introduced in this paper have been explored initially in [3] that identified a new approach to software benchmarking and estimation.

The mathematical estimation models from the literature are broadly derived from two distinct strategies that take into account information from completed projects:

- Strategy 1: Statistical analyses represented by multi-variable models with as many independent variables as the cost drivers taken into account. Some examples are linear and nonlinear regressions techniques, neural network models, and genetic algorithms [4, 5]. For an adequate statistical analysis, it is generally accepted that there should be 20 to 30 observations for each independent quantitative variable.
- Strategy 2: Statistical analyses with a unique independent variable (typically size) combined with a single adjustment that combines the impact of multiple cost drivers, individual values of which come from fixed pre-determined step-functions for each cost driver. This can be observed, for instance, in COCOMO-like models [6, 7].

Multi variables models built with insufficient data points (as in strategy 1) or with models with an adjustment factor bundling multiple categorical variables (strategy 2) do not necessarily reduce the risks inherent in estimation. They may lead managers to believe that the majority of important cost drivers have been duly taken into account by the models whereas, in practice, even more uncertainty may have been created [8, 9]. Numerous other mathematical techniques exist in software engineering, such as analogy-based reasoning and machine learning estimation models, which differ from the above in their mathematical peculiarities, but which similarly use a multi-variable approach [10, 11].

Although accurate estimation of a single project is important, estimation is not the unique management concern, nor the most important one for a specific project or for a set of projects for an organization or a customer. For example, greater productivity, profitability, and high quality have often greater management relevance than accuracy of estimation.

Many current estimation models are built without reference to productivity issues, frequently taking into account a large number of variables (at times over too small data sets) in an attempt to predict the better fit of data points, and then evaluating these models by how

close the models 'estimates' are to 'actual data'. However, when actual (internal or external) data come from highly unproductive projects (and uncompetitive ones) plagued with numerous quality issues, that an estimation model estimate 'accurately' is of limited value to customers.

Similarly, how relevant are estimation models built from external data when a software organization cannot compare its own productivity with the productivity of the organizations the data is coming from? If its own productivity is lower, an external-based estimation technique will under estimate: without insights into its own productivity, use of external-based estimation tools is an inadequate approach.

The rest of the paper is organized as follows. Section II presents the productivity concept as defined in economics to represent the performance of a production process, including fixed/variable costs and economies/diseconomies of scale. This section also illustrates how to recognize these concepts in software engineering data sets. Section III presents how these concepts are used in a Canadian organization to address management requests for information on the productivity of their development process. Section IV presents the productivity analysis and the estimation models developed for the organization on the basis of economic concepts. Section V presents the usage of the SWEBOK Guide to identify process improvement opportunities for the organization. Section VI presents a summary and implications for estimation effort.

II. PRODUCTIVITY MODELS AND ECONOMICS CONCEPTS

A. A productivity model represents a 'production' process

A project is typically set up to plan and manage a unique event, with a start date, an end date, and a unique outcome that typically has not been produced before. Building a house is a project, building a road is a project, as is developing a software application A for customer B for a specific deadline.

To improve the odds of meeting the project targets a project process is implemented to plan activities, monitor project progress and take remedial action when something goes off track. Similarly, even though each piece of software is different, its delivery is organized in a structured manner and not left to randomness and individual moods and intuitions of the day. To deliver the right outcome on time and within the expected cost and level of quality, a 'development process' is implemented to meet the target taking into account the set of priorities within a reasonable range of predictability.

A project, including software projects, is a process and each process corresponds to a level of performance aligned with its own specificities in terms of activities, structure of activities, constraints and resources involved in the process. The question is: How can the performance of a process be estimated in the future if its current and past performance and any variations in performance are not known? What are the economic concepts at work in software projects? And, when this is understood and quantified, how can these economics insights be used for estimation purposes?

A software development project can be modeled as a production process, in its simplest form using three main components:

1) Inputs: to calculate productivity, the people involved in the production process are considered as the inputs from an economics perspective. In a software project, the inputs are typically measured in work-hours (or person-days/-weeks/-months).
2) Activities within the process itself: for calculating productivity, all of the activities and constraints of the process are considered as a black-box and are not taken into account: they are, therefore, implicit variables, not explicit variables in productivity calculations.
3) Outputs: the outputs are represented by the number of functional units produced by the process. The output of the software development process is the set of functions delivered to the users, which functions can now be quantified with international standards of measurements, such as with any of the relevant ISO standards on software functional size [12-15].

The productivity of a process is its ratio of outputs over the inputs used to produce such output. In software, the productivity of a software project can be represented, for example, as 10 Function Points per work-month. It is to be observed as well that, by convention, the productivity ratio ignores all process characteristics: it is process and technology independent and, therefore, allows objective comparison of the productivity of a process across technologies, organizations and time.

Productivity describes this single concept, and does not explain why the productivity has varied and may vary over time within the same process, or across distinct processes. To explain productivity variability (within and across processes) additional variables are necessary. Multi-variable models are useful to investigate which variable impacts productivity (in a positive or negative manner), and to what extent. The investigation of why the productivity of a process varies is the realm of efficiency studies, not productivity studies.

B. Productivity models with fixed and variable costs

The use of productivity models has a long history that can be traced back to a large body of knowledge developed in the domains of economics and engineering. This section introduces some of these concepts, which may also be useful in modeling, analyzing and estimating the performance of software projects.

A productivity model is typically built with data from completed projects, that is, it uses the information of a project for which there is no more uncertainty:

- The outputs: i.e., all the software functions have been delivered; and,
- The hours worked on the project: i.e., they have been accurately entered into a time reporting system.

This is illustrated in Fig. 1 where:

- The x axis represents the functional size of the software projects completed;
- The y axis represents the effort in number of hours that it took to deliver a software project.

The straight line across Fig. 1 represents a statistical model of the productivity of the software projects. More specifically, this single independent variable linear regression model represents the relationship between effort and size, and is represented by the following formula:

Y (effort in hours) = f(size)

= a x Size + b where:

- Size = number of Function Points (FP)
- a = variable cost = number of hours per function point (hours/FP)
- b = constant representing fixed cost in hours

In terms of units, this equation gives:

Y (hours) = (hours/FP) x FP + hours = hours

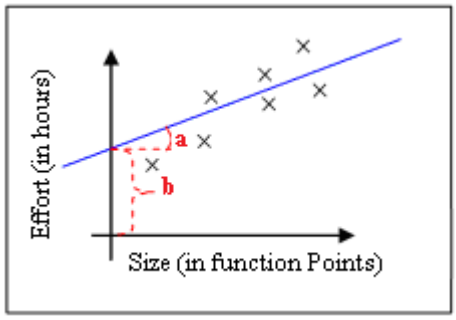


Figure 1: Fixed & variable cost in a productivity model

Insights from economics have identified two distinct types of costs incurred to produce different quantities of the same types of outputs:

Fixed costs: the portion of the resources expended (i.e., inputs) that does not vary with an increase in the number of outputs. In Fig. 1, this corresponds to b, the constant in hours at the origin when size = 0.

Example of a *fixed* cost: a cost of b hours of project effort is required for mandatory project management activities, whatever the size of the software to be developed.

Variable costs: the portion of the resources expended (i.e., inputs) that depends directly on the number of outputs produced. In Fig. 1, this corresponds to the slope of the model, that is: slope = a in terms of hours/FP (i.e., the number of work hours required to produce an additional unit of output).

It is to be observed that in productivity models, the constant b does not represent the errors in the estimates as in multi-variable estimation models. In productivity models, b has a practical interpretation corresponding to the economic concepts explained above, that is: the portion of the cost that does not vary with increases in the production outputs.

C. *Wedge-shaped datasets in software engineering*

Often, a graphical representation of projects in large datasets has the wedge-shaped distribution illustrated in Fig. 2 with the software size as the single independent variable. It can be observed in Fig. 2 that, as the project size increases on the x axis, there is a correspondingly larger dispersion of the data points across the vertical axi. In other words, there are increasingly wide variations in project effort on the y axis as the project size increases, that is, large productivity differences across software of similar size delivered. Such wedge-shaped datasets have initially been observed by [16, 17] and are representative of datasets collected from multiple organizations, each with their own distinct development processes using a variety of technologies and corresponding distinct abilities to exploit them.

Looked at from a control process view point within a single organization, a wedge-shape data set could represent:

A. A process 'out of control,' that is, a process with a large variation in productivity across increases in the size of the outputs is due to a lack of repeatability in a process, i.e., an ad-hoc process dependent on individual actions and expertise and unknown quality, rather than repeatable and 'under control,' as happens when a development methodology is enforced, leading to repeatability However, a process 'under control' may be highly repeatable, predictable and with high quality, but it may concurrently be highly inefficient and expensive; or,

B. Data originating from various distinct processes, each with their distinct productivity ratios, thereby 'only appearing as out of control' because the single-variable model does not take into account the presence of the distinct processes of each organization, each with their distinct productivity ratios; or

C. A process where each project has large variations in unit-effort due to factors other than size, that is a multi-variable dependent process. Adequate modeling of such factors in multi-variable models is only feasible when there are enough data points (i.e., the sample size should increase by 20 to 30 projects for each additional variable introduced in the model). Models introducing multi-variables without sufficient data points will provide mathematical models with

quantitative parameters, but will have no generalization power for future usage, including in similar contexts.

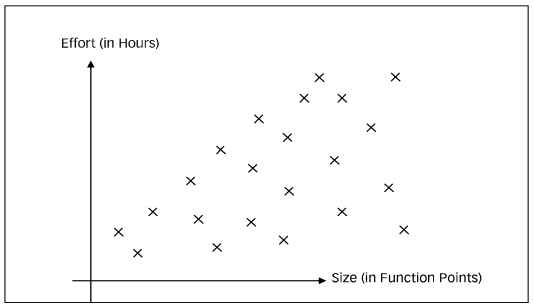


Figure 2: Example of a wedge-shaped productivity dataset

It is obvious from Fig. 2 that a single-variable productivity model built from this data set is not directly useful for estimation purposes in this context. All other process and product variables combined together have a large impact on the total variation of the dependent variable (i.e., here, Effort). Therefore, for estimation purposes, multi-variable models would be necessary. However, such multi-variable models, while useful for estimation purposes, do not allow productivity comparisons and evaluation.

The next question is: what causes these different behaviors? Of course, the answers cannot be found by graphical analysis alone, since in the productivity model there is only a single independent quantitative variable in a two-dimensional graph. This single independent variable does not provide, by itself, any information about the other variables, or about similar or distinct characteristics of the completed projects for which data are available. Efficiency investigation with additional independent variable can help identify which other variables cause variations in productivity and to what extent for each.

When a data set is large enough (that is 20 to 30 data points for each independent variable), the impact of the other variables can be analyzed by statistical methods. In practice, most software organizations do not have data set large enough for valid multi-variable statistical analysis. However, within a single organization the projects included within a data set can be identified nominally by the organizations that collected the data [3, 16]. Each project in each subset should be subsequently analyzed to determine:

- Which of their characteristics (or cost drivers) have similar values within the same subset; and
- Which characteristics have very dissimilar values

across the two (or three) subsets.

Of course, some of these values can be descriptive variables with categories (i.e., on a 'nominal' scale type: for example, a specific Data Base Management System (DBMS) has been used for a subset of projects, etc.). It then becomes necessary to discover which additional independent variables have the most impact on the relationship with project effort. The different values of such characteristics can then be used to characterize such datasets, and set the parameters for selecting which of these productivity models to use later on for estimation purposes.

D. Homogeneous datasets in software engineering

Another type of project distribution is represented in Fig. 3, which illustrates a strong consistency in the dispersion of the effort as size increases. This would represent more homogeneous data sets in which the increase in software size explains well the increase in effort. Such a homogeneous distribution of software projects data appears as well in the literature [18-21]. In these datasets, the increase in functional size explains 80% to 90% of the increase in effort, while all of the other factors together explain at most 10% to 20% of the increase in effort. Such datasets would be considered homogeneous with respect to the dependent and independent variables being investigated. This low dispersion in project productivity would typically have one or a mix of the following causes:

- The project data comes from a single organization with well implemented development standards.
- The project data is representing the development of software products with very similar characteristics in terms of software domains, non-functional requirements and other characteristics.
- The development process is under control with a predictable productivity performance.
- Data collected in an organization based on an in-process sound measurement program, and where standardized measurement definitions have been adopted by all projects participants, leading to high data integrity.

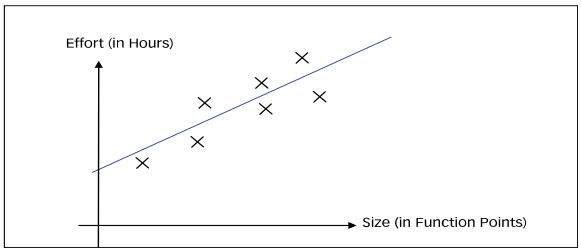


Figure 3: A homogeneous productivity dataset

It is obvious from Fig. 3 that the one-variable productivity model built from this data set is directly useful for estimation purposes. All other process variables combined together have a very small impact on the total variation of the dependent variable (i.e., here, Effort). Looked at from a control process view point, this data set represents a process 'under control', that is a process with

a predictable performance, both in terms of productivity and efficiency.

III. A PRACTICAL USE OF THESE ECONOMIC CONCEPTS: AN EMPIRICAL STUDY

A. Context

A Canadian organization interested in determining its own productivity, in understanding some of the key drivers behind its major productivity variations, and in using the findings to improve its organizational performance in general and its estimation process in particular was selected for the empirical study.

This organization, a government agency, provides specialized financial services to the public, and its software applications are similar to those of banking and insurance providers. It has a software development methodology fully implemented across all of its projects. The main objectives of the empirical study were:

1. Internal benchmarking, i.e., compare the productivity of individual projects.
2. Develop estimation model(s) based on the data collected.
3. Identify and explain significant productivity variations across the organization's projects.
4. Identify opportunities for process improvement.

B. Data collection procedures

The initial step was to identify projects that could be measured for the productivity and benchmarking analyses. The selection criteria were:

- Projects completed within the previous two years, and
- Project documentation available for functional size measurement.

For this study, all data were recorded using the data field definitions of the data collection questionnaire of the International Software Benchmarking Standards Group [22].

C. Data Quality Controls

Quality control of the data collection process is important for any productivity study. Here, two quantitative variables are critical: the effort reported for each project, and the project functional size:

A- Effort data: in this organization, the time reporting system is considered highly reliable and is used for decision making, including payment of invoices when external resources are hired to complement project staffing.

B- Measurement of functional size: the quality of the measurement results depends on the expertise of the measurers and on the quality of the documentation available for the measurement process. For this productivity study: all functional size measurements were carried out by the same measurer with 20 years expertise in functional size measurement.

Table I from [23] lists the criteria used to rank the quality of the documentation used for measuring functional size, on the basis of the documentation of the individual functional processes developed [23, 24]. Note that this is not a global subjective assessment of the documentation, but an assessment of the documentation of each of the functional processes based on the detailed documentation elements available for measurement. The individual rankings of each functional process were recorded by the measurer in parallel to the measurement of the functional size of each of the 16 projects.

Table I: Criteria for ranking documentation quality [23]

Rank	Criteria
A	Every function completely documented
B	Function documented, but without a precise data model
C	Functions identified at a high level, but without any detail
D	An approximation of the number of functions is available, with the individual functions not listed
E	Some functions are not explicitly described in the documentation, but an expert measurer adds information based on his expertise, e.g., missing validation functions

Table II reports the documentation quality rankings of each project and specifies for each project what proportion of the project documentation met the various criteria listed in Table I. For this study, the documentation is considered 'good' when it meets criterion A or B in Table I. The following observations were made from Table II:

- For 11 projects: the documentation of more than 95% of the functional processes measured was rated as being of good quality (equal to A or B). Considering the extended measurement expertise of the measurer and the high quality of documentation, the size measured for these 11 projects can be considered highly accurate.
- For Projects 3 and 13: the documentation quality was rated as being of good quality for 62% and 71% of the functional processes respectively.
- For project 10: the documentation was rated as being of average quality (criterion C). This could impact the accuracy of the size measured for this project, as it used less detailed documentation.
- The documentation of Project 7 was rated as being of good quality for 31% of the functional processes.
- For Project 8: most of the functions measured had to be derived from documentation at a very high level, that is criterion E = 100%. This means that the

functional size for this project has been approximated rather than measured precisely, with an undetermined range of size variation.

Overall, at the detailed level, 85% of all the processes measured for the 16 projects had a good level of documentation and provided a sound basis for the measurement of the functional size of the projects included in the productivity analysis reported in this paper.

Table II: Quality of the documentation ranked at the functional process level

Project Id.	Distribution of the quality ranking at the functional process level				
	A	B	C	D	E
1	11%	85%	4%		
2		100%			
3	54%	42%	2%	2%	
4		68%	38%		
5		100%			
6		100%			
7		100%			
8		31%	69%		
9					100%
10		100%			
11			74%		26%
12	17%	83%			
13		71%	29%		
14	76%	24%			
15		100%			
16		100%			

D. Descriptive Analysis

1) Projects characteristics

For this study, the 16 software development and improvement projects were measured in terms of functional size, effort, and various environment qualifiers. The staff who developed these projects included both internal and external developers, distributed equally overall. In this dataset:

- Project size varied from a minimum of 111 FP to a maximum of 646 FP.
- Effort varied from 4,879 hours to 29,246 hours.
- Unit effort varies from 14 h/FP 12 to up to 98 h/FP, a factor of approximately eight between the least productive and the most productive within the same organization.
- Duration varied from 10 to 35 months.
- Maximum development team size for 12 of the 16 projects ranged from 6 to 35 employees.

The descriptive statistics of this dataset are as follows, while details are reported in [8 – Table 12.2]:

- Average project effort = 12,033 hours (or, 1,718 person-days at 7 hours per day, or 82 person-months at 21 days per month).
- Average unit effort = 41.5 h/FP
- Average project duration = 18 calendar months.
- One-third of the projects = newly developed software.
- Two-thirds of the projects = functional enhancements to existing software.

2) Project priorities

Each project typically met four targets that are determined at project inception:

- Project scope (i.e., the functions to be delivered to the users)
- Project cost (or effort)
- Project deadline
- Quality of software delivered

In a context of limited resources and a high level of uncertainty, it is extremely challenging to meet all these targets at the same time, and a number of compromises must be made during a project life cycle. To empower project managers to make such compromises, an organization will typically determine the priority targets for each project among those specific to any one project. The information on the priorities assigned to each of these four targets was collected through interviews of the project managers of these projects and recorded during the data collection process. A summary of project priorities is presented in Table III, where:

- For 8 of the 16 of the projects (i.e., 50%), the 'deadline' target was listed as priority 1.
- For 75% of the projects, the 'scope' target was listed as priority 1 or 2.
- For 50% of the projects, 'quality' was listed as priority 2 (and two projects listed it as priority 1).
- For none (0%) of the projects was the 'cost' target listed as priority 1 (and only one project listed cost as priority 2).

Table III: Summary of project priorities

	Number of projects			
	Priority 1	Priority 2	Priority 3	Priority 4
Deadline	8	1	1	6
Scope	6	6	4	0
Quality	2	8	2	4
Cost	0	1	9	6

These observations indicate that for this organization the project's deadline is a high priority, while cost is a low priority. The reasons for favoring the deadline over the cost of the measured project were the following:

- The urgency to solve the problem for internal reasons;
- Obligations linked to current laws or future ones;
- The pressures of client managers for the reasons mentioned above or other reasons.

IV. PRODUCTIVITY ANALYSIS AND ESTIMATION MODELS

A. *The overall productivity model for the organization*

The dispersion of points for the organization is illustrated in Fig. 4 for all 16 projects, with functional size on the x axis, and effort on the y axis: at first sight it appears somewhat like a wedge-shape data set rather than an homogeneous data set with respect to the functional size at the independent variable.

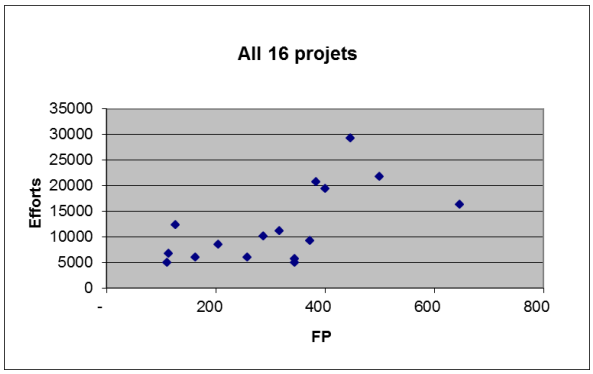


Figure 4: The dispersion of project productivity for the organization – N = 16 projects [8]

Fig. 5 shows next the overall single-variable productivity model for the organization, using a single regression model:

Effort = 30.7 h/FP x project size + 2,411 h (Fig. 5)

The coefficient of determination (R^2) of this model is relatively low, at 0.39.

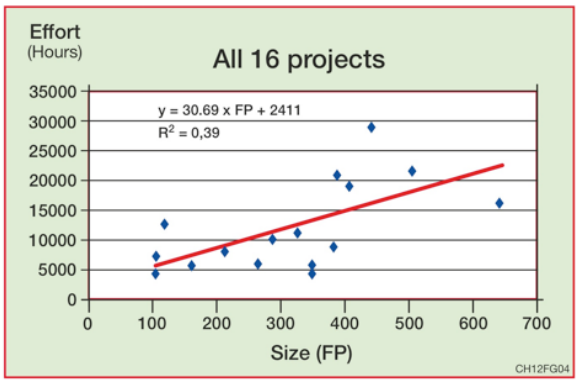


Figure 5: The organization's overall productivity model – N = 16 projects [1]

The practical interpretation of the above equation is as follows:

- Fixed effort = 2,411 h
- Variable effort = 30.7 h/FP

Possible reasons for the high fixed and high variable unit effort numbers discussed with the managers, and the following observations provided in terms of the development methodology deployed in the organization:

A. It is highly procedural and time-consuming.
B. It included heavy documentation requirements.
C. It required lengthy consensus building procedures across stakeholders and development staff.
D. It required a relatively high number of inspections.

This is the productivity model that should be used, across the 2-year time period, for later internal benchmarking as well as for external benchmarking. Productivity models based only on functional size allow independent comparisons across time periods and variations of technologies of processes used by others.

From Fig.5, it can be observed that, for this organization, five projects had effort 100% higher than projects of comparable functional size:

- A project of approximately 100 FP required twice as much effort as two other projects of similar size.
- Four large projects (between 400 and 500 FP) required two or three times more effort than, projects only relatively smaller (between 350 and 400 FP). The effect of these four projects was to pull up the linear model (and corresponding slope) and considerably influence both the fixed and variable costs.

Therefore, this data sample was split into two groups for further analysis.

A. The group of 11 projects that have the best productivity (i.e., lower unit effort, and that are below or very close to the regression line in Fig. 5).
B. The group of five projects that have a productivity much worse (i.e., a unit effort twice the unit effort of the 11 other projects, and that are largely above the regression line in Fig. 5).

B. *Organizational process capability: the most productive projects*

Fig. 6 presents the 11 projects with a much lower unit effort per project that is, those which were most productive. For these projects, the linear regression model is:

Effort = 17.1 h/FP x size of the project + 3,208 h

The coefficient of determination (R^2) of this model is 0.589, higher, relatively, than that for the overall model.

The practical interpretation of this equation is:

- Fixed costs = 3,208 h
- Variable Costs = 17.1 h/FP

C. *Productivity model of the least productive projects*

For the five least productive projects in group B, the productivity model in Fig. 7 is:

Effort = 33.4 h/FP x project size + 8,257 h

The coefficient of determination (R^2) of this model is better, at 0.637. Of course, with a sample of only five projects, this number is not statistically significant, but is still interesting for the organization.

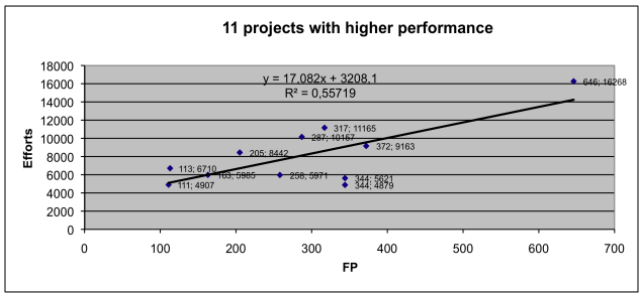


Figure 6: Most productive projects - N = 11 projects [8]

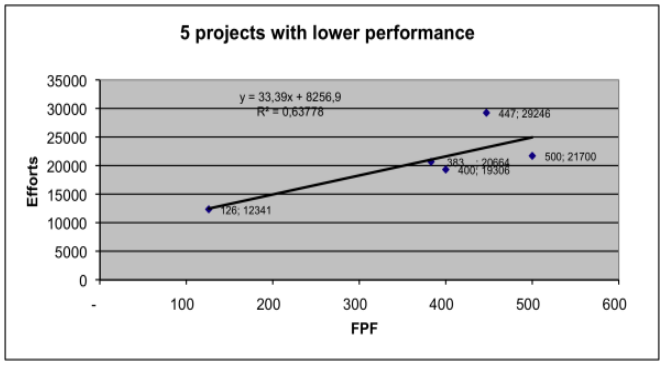


Figure 7: The productivity model of the least productive projects – N = 5 [8]

The practical interpretation of the above equation is as follows:

- Fixed effort = 8.257 h
- Variable effort = 33.4 h/FP

This group of the five least productive projects is characterized by a fixed cost that is almost four times higher than that of the full set of projects (8,257 hours vs. 2,411 hours), and a relatively similar variable effort unit (33.4 h/FP vs. 30.7 h/PF).

The group of 11 most productive projects is characterized by a fixed cost approximately 40% lower than that of the least productive projects (3208 hours vs. 8257 hours), and a variable unit effort almost 50% lower (17.1 h/FP vs. 32.4 h/FP); that is, with interesting economies of scale and an R^2 of 0.55.

A summary of each group is presented in Table IV, where 11 projects represent what the organization is 'capable' of delivering in normal conditions and the other five projects illustrate how projects are significantly impacted by the presence of factors that have not yet been identified through the single independent variable (i.e., functional size) analysis. Exploration of these additional impact factors is discussed in the next sub-section D.

The single variable productivity model still provides useful insights to the organization and allows it to monitor its own productivity models across time. For example:

A. Would the fixed-variable cost improve over the next two-year period? A subsidiary question would be to identify the factors that caused this positive (or 'negative') impact?
B. Could the organization avoid or, when avoidance is not possible, more effectively mitigate, the occurrence of the negative factors beyond the project manager's control?

Table IV: Fixed & variable efforts: Capability versus least productive projects [1]

Samples/ Regression coefficients	All 16 projects	Most productive: 11 projects	Least productive: 5 projects
Fixed effort (hours)	2,411	3,208	8,257
Variable effort (h/FP)	30.7	17.1	34.4

D. Qualitative causal analysis

The data collected has allowed us to identify the overall productivity of the organization, to observe the fixed and variable contributions of its process capability, as well as observing an 100% increase in both fixed and variable costs when there were factors negatively impacting on the productivity.

Of course a single independent variable model cannot explain the causes of such variations. Furthermore, with a dataset of only 16 projects, there are not enough data points within a single organization (unless they have been collecting data for many years) to rely on quantitative analysis. Each additional independent typically requires 20 to 30 additional data points. In the absence of sample sizes large enough for quantitative analysis, qualitative analysis can help identify probable causes of increases. In the context here, qualitative analysis will not attempt to quantify precisely the impact of a cause (or cost drivers), but will attempt to identify qualitatively factors that could have the greatest negative impact on productivity.

Firstly, in the causal analysis of the productivity variations in the organization, we eliminated two candidate cost drivers since they were considered constant in both groups of productivity performance:

- Development methodology: in the organization the use of the industry-tailored development methodology was fully deployed across all software development projects: none of the activities and controls was bypassed. Therefore, there was no development methodology difference across all projects.
- Project management expertise: some of the projects managers had, within this same two-year period, supervised projects that were both among the most productive and the least productive. Therefore, the expertise of specific project managers could not explain large project productivity differences.

The question that arises: What are the factors that led to such large (i.e., +100%) increases in unit effort? What may be the major cause-effect relationships? To identify and investigate these relationships, available project managers were interviewed to obtain feedback as what they believed had contributed to either an increase or a decrease in the productivity of their respective projects. The project managers interviewed had managed seven of the 16 projects:

A. Three projects with the lowest productivity (i.e., the highest unit effort);
B. Two projects with average productivity;
C. Two projects with the highest productivity (i.e., the lowest unit effort).

The aim of the interviews was to obtain qualitative information from the project managers on the factors they believed had contributed, or not, to the increase in project effort compared to that of other projects of similar size developed in the organization's environment or elsewhere during their project management practice. Their feedback is summarized as follows:

A- The most productive projects had the following characteristics:

1. Users familiar were with both the business and software development processes;
2. Users were involved throughout the project;
3. Software developers working on the projects were experienced in the use of the development environment.

B. The least productive projects had the following characteristics:

B1. Customer related issues:

1. Customer requirements were poorly expressed, or a customer representative did not know his environment (business area), leading to frequent change requests during a project life cycle.
2. High turnover of users involved in the projects, leading to instability in the requirements and delays in decision making.
3. Customers not familiar with the software development process in the organization, including their required involvement in project activities, including activity reviews.

B2. Project constraints:

1. Tight project deadlines for legal constraints or public face-saving leading to compressed scheduling and resources 'piled up' to make the problem disappear.
2. New technologies unknown to the developers.

B3: Product constraints: Multiple links with other software applications of the organization.

An example of a negative product constraint was reported for the project with the highest unit effort (98 h/FP). The software delivered by this project was of a small functional size, but required twice as much effort to develop as another of similar size as it interacted with almost all the other software applications of the organization and was dependent on other organizational units. Another project had a very tight deadline, which led management to 'throw' resources at the problem to meet the deadline irrespective of the total effort required.

It can be observed that although it was possible to identify 'qualitatively' some factors with major negative impact, the sample size was much too small for statistical tests to quantify such impact.

V. IDENTIFICATION OF PROCESS IMPROVEMENTS USING THE SWEBOK GUIDE

A. Implementation and coverage of best practices

In previous years, the organization had invested considerably in designing and deploying improvements to its development methodology. For this empirical study, additional analyses were carried out to identify process strengths and weaknesses in order to identify improvement opportunities in development processes and techniques.

The study included verification of the use of the recommended best software engineering practices in the participating organization. For this verification, the software engineering practices listed in the SWEBOK Guide [2] were used. This guide represents a broad international consensus on the concepts and practices of software engineering that are recognized as providing benefits to the majority of projects in most cases. All 10 knowledge areas of the 2004 version of the SWEBOK Guide were taken into account, with the exception of the maintenance knowledge area, which was not relevant for this empirical study – see Table V.

To collect this information, the quality specialist participating in all project phases for each project was interviewed and asked to confirm whether or not each of the practices in the SWEBOK Guide was indeed being widely used across all projects in the organization.

B. Coverage of best practices

The percentages of software engineering practices in use in the organization for each knowledge area are presented in Table VI, in decreasing order of coverage. These percentages represent the ratio between the practices observed to be in general use in this organization, divided by the total number of practices listed in each of the SWEBOK knowledge areas.

From all the knowledge areas related directly to the development life cycle, that is, from requirements engineering up to software testing, those with over 60% coverage indicates a very widespread use of organizational processes across the organization. In comparison to the capability level of the CMMI model, this would align with a number of Key Process Areas at Level 3, meaning that the software engineering processes

in the organization have been deployed and are in use throughout the organization.

Table V: SWEBOK 2004 Knowledge Areas

SWEBOK Knowledge Areas
Requirements Engineering
Software Design
Software Construction
Software Testing
Software Maintenance
Software Engineering Management
Configuration Management
Software Engineering Process
Software Quality
Tools and Methods

Table VI: Coverage of SWEBOK practices

SWEBOK Knowledge Areas	% coverage
Software Design	89
Software Management	75
Configuration Management	74
Requirements Engineering	73
Software Construction	71
Software Testing	61
Tools & Methods	54
Software Engineering Process	44
Software Quality	38

However, coverage was much less extensive in support areas, such as Tools & Methods, Software Engineering Process, and Software Quality, where the product and process measures were not covered at all. For this reason, the organization had no measurable information on the effectiveness of the implementation of their practices and the benefits derived, that is, quantitative information to support the decision making process was lacking.

This can be illustrated in the following way: while each project must use the corporate development process it generally lacked data for its evaluation and control functions. Therefore, the organization would not qualify for the fourth level of capability using the CMMI process evaluation model.

VI. SUMMARY AND IMPLICATIONS FOR MANAGEMENT

This paper has reported on productivity analysis of software projects developed by a governmental organization utilizing the productivity concepts from the field of economics, including fixed/variable costs modeling. For the organization studied, three productivity models were identified that represented respectively:

- An overall productivity model of the organization that can be used for external benchmarking purposes. This overall productivity model can be used later across other times periods to verify whether or not productivity of the organization is improving over time, and with respect to external similar organizations.
- A productivity model built from the best productive projects representing a capability to deliver a software project with a lower fixed/variable effort structure, in the absence of major disruptive factors.
- A productivity model based on the five projects with the highest unit effort: in this case, the presence of disruptive factors led to greater than 100% increase in project effort in comparison to the organization's capability for process productivity.

Of course, the limited number of projects available in these mathematical models does not permit generalization to other contexts, but does describe quantitatively and objectively many features of productivity in the organization. These models are representative of the organization studied where a unique software development methodology is widely implemented and represents well deployed, repeatable corporate software practices, rather than unpredictable individual and ad-hoc practices.

For future project estimation, the organization should use the process capability model represented by the best performing projects, provided that a risk analysis has not detected the presence of any of the disruptive factors that have in the past increased effort twofold. Whenever such disruptive factors are identified with a high probability of occurrence the organization should estimate such projects using the productivity model derived from the least productive projects. The use of these two single-variable productivity models would be expected to provide more accurate estimates than the overall productivity model combining all previous projects.

An organization such as the one studied having measured only a small set of projects is typical of many organizations without much historical data: there are not enough data points to build with high confidence multi-variable estimation models representing local conditions and related organizational performance.

The insights from productivity models developed from an economic perspective are important since relevant improvement activity may directly impact the productivity of the organization, by lowering either the fixed or variable project costs.

Furthermore, the empirical study also identified opportunities for improvement in three areas, namely:

1. Early identification of project risks with a potentially twofold impact on project effort.
2. Increase in project management efficiency:
   - Improvement in productivity analysis and current productivity models.
   - Improvement in the estimation process.
3. Process improvement:

- Establishment of mechanisms for monitoring and evaluating processes.
- Reduction in fixed effort (establishment of predefined selection rules in the project context).
- Monitoring the impact of new technologies and new development processes.

REFERENCES

[1] A. Abran, J. M. Desharnais, M. Zarour, and O. Demirors, "Productivity Based Software Estimation Model: An Economics Perspective and an Empirical Study," 9th International Conference on Software Engineering Advances - ICSEA 2014, Nice (France), pp. 196-201, 2014.

[2] Abran A. and Moore, J. (co-executive editors), Bourque, P. and Dupuis, R. (co-editors), Tripp, L. (2005), "Guide to the Software Engineering Body of Knowledge - 2004 Version - SWEBOK," IEEE-Computer Society Press, 200 pages.

[3] A. Abran and J. J. Cuadrado, "Software Estimation Models & Economies of Scale," presented at the 21st International Conference on Software Engineering and Knowledge Engineering - SEKE'2009, Boston (USA), July 1-3, 2009.

[4] M. Jørgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Transactions on Software Engineering*, vol. 33 no. 1, pp. 33-53, 2007.

[5] M. Shepperd and M. and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology, Elsevier*, vol. 54, pp. 820–827, 2012.

[6] B. Boehm, *Software Engineering Economics*: Englewood Cliffs, NJ, Prentice Hall, 1981.

[7] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. Clark, E. Horowitz, *et al.*, *Software Cost Estimation with COCOMO II*: Prentice Hall, 2000.

[8] A. Abran, *Software Project Estimation – The Fundamentals for Providing High Quality Information to Decision Makers*: IEEE-CS Press & John Wiley & Sons – Hoboken, New Jersey, 2015.

[9] Barbara Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *5th International Conference on Predictor Models in Software Engineering, PROMISE*, Vancouver, BC, Canada, 2009.

[10] F. A. Amazal, A. Idri, and A. Abran, "Analogy-based Software Development Effort Estimation: A Systematic Mapping and Review," *Information and Software Technology*, vol. 58, pp. 206-230., 2014.

[11] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology, Elsevier*, vol. 54, pp. 41-59, 2012.

[12] International Organization for Standardization (ISO), "ISO/IEC 20968: Software Engineering - Mk II Function Point Analysis - Counting Practices Manual," ed. International Organization for Standardization, Geneva, 2002.

[13] International Organization for Standardization (ISO), "ISO/IEC 24750: Software Engineering - NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis," ed. International Organization for Standardization, Geneva, 2005.

[14] International Organization for Standardization (ISO), "ISO/IEC 20926: Software Engineering - IFPUG 4.1 Unadjusted functional size measurement method - Counting Practices Manual," ed. International Organization for Standardization, Geneva, 2009.

[15] International Organization for Standardization (ISO), "ISO/IEC 19761: Software Engineering – COSMIC - A Functional Size Measurement Method," ed. International Organization for Standardization, Geneva, 2011.

[16] B. Kitchenham, "Empirical studies of assumptions that underlie software cost-estimation models," *Information and Software Technology*, vol. 34, pp. 211-218, 1992.

[17] B. Kitchenham and N. Taylor, "Software Cost Models," *ICL Technical Journal*, vol. 4, pp. 73-102, 1984.

[18] K. Lind and R. Heldal, "Estimation of Real-Time Software Code Size using COSMIC FSM," presented at the IEEE Intl. Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC), Tokyo, Japan, 2009.

[19] K. Lind and R. Heldal, "A Model-Based and Automated Approach to Size Estimation of Embedded Software Components," presented at the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems, Wellington, New Zealand, 2011.

[20] S. Stern and O. Guetta, "Manage the automotive embedded software development cost by using a Functional Size Measurement Method (COSMIC)," presented at the 5th International Congress and Exhibition: Embedded Real Time Software and Systems, ERTS, Toulouse, France, 2010.

[21] A. Abran and P. Robillard, "Function Points Analysis: An Empirical Study of its Measurement Processes," *IEEE Transactions on Software Engineering*, vol. 22, pp. 895-909, 1996.

[22] ISBSG R11. (2009, 2015.01.15). *International Software Benchmarking Standard Group*. Available: http://www.isbsg.org/

[23] J. M. Desharnais and A. Abran, "Quality of Functional User Requirements documentation using COSMIC ISO 19761: verification process," presented at the International Workshop on Software Measurement – IWSM Stuttgart, Germany, 2010.

[24] J. M. Desharnais, B. Kocatürk, and A. Abran, "Using the COSMIC Method to Evaluate the Quality of the Documentation of Agile User Stories," presented at the 21st International Workshop on Software Measurement – 6th International Conference on Software Process and Product Measurement - IWSM-Mensura, Nara, Japan, 2011.

An Extensible Benchmark and Tooling for Comparing Reverse Engineering Approaches

David Cutting and Joost Noppen

School of Computing Science
University of East Anglia
Norwich, Norfolk, UK
Email: {david.cutting,j.noppen}@uea.ac.uk

***Abstract*—Various tools exist to reverse engineer software source code and generate design information, such as UML projections. Each has specific strengths and weaknesses, however, no standardised benchmark exists that can be used to evaluate and compare their performance and effectiveness in a systematic manner. To facilitate such comparison in this paper we introduce the Reverse Engineering to Design Benchmark (RED-BM), which consists of a comprehensive set of Java-based targets for reverse engineering and a formal set of performance measures with which tools and approaches can be analysed and ranked. When used to evaluate 12 industry standard tools performance figures range from 8.82% to 100% demonstrating the ability of the benchmark to differentiate between tools. To aid the comparison, analysis and further use of reverse engineering XMI output we have developed a parser which can interpret the XMI output format of the most commonly used reverse engineering applications, and is used in a number of tools.**

Keywords–Reverse Engineering; Benchmarking; Tool Comparison; Tool Support; Extensible Methods; XMI; Software Comprehension; UML; UML Reconstruction.

I. INTRODUCTION

Reverse engineering is concerned with aiding the comprehensibility and understanding of existing software systems. With ever growing numbers of valuable but poorly documented legacy codebases within organisations reverse engineering has become increasingly important. As explored in our previous work [1], there are a wide number of reverse engineering techniques, which offer a variety in their focus from Unified Modelling Language (UML) projection to specific pattern recognition [2][3][4][5].

However, it is difficult to compare the effectiveness of reverse engineering techniques against each other, as no standard set of targets exist to support this goal over multiple approaches, a problem also found in the verification and validation of new tools and techniques [6]. Any performance evaluations that do exist are specific to an approach or technique. Therefore, it is impossible to gain a comparative understanding of performance for a range of tasks, or to validate new techniques or approaches. Therefore, this paper introduces a benchmark of such targets, the Reverse Engineering to Design Benchmark (RED-BM), created in order to compare and validate existing and new tools for reverse engineering.

Therefore, our goals are to:

- Create a benchmark suitable for empirical comparison of reverse engineering tools
- Apply this benchmark to current tools and evaluate the result, and their performance
- Make the benchmark extensible to support further approaches
- Provide support for extensibility through the means of data exchange between implementations

The benchmark described in this article builds on and extends previous work [1], including greater details and specifics on an inherent extensibility mechanism with an example.

The intent of the benchmark, in addition to ranking of current reverse engineering approaches, is to support further extensibility in a generic and easily accessible manner. To achieve this, a number of tools have been developed and provided which aid in the open analysis and exchange of reverse engineering output.

The remainder of this paper is organised as follows: in Section II, we introduce our benchmark before introducing the target artefacts (Section II-A) provided. Section II-B covers the performance measurements used, with Section II-C detailing how complexity is broken down for granular measurement. Extensible features of the benchmark are demonstrated in Section III, specifically the definition of new measurements (Section III-A) and use of reverse engineering output for data exchange (Section III-B). Section IV details the toolchain support for the benchmark. In Section V, the benchmark is applied against a number of industry standard tools with an evaluation of these results in Section VI and a discussion in Section VII. Related work is covered in Section VIII and the final Section IX draws a conclusion and identifies our future direction for research.

II. THE REVERSE ENGINEERING TO DESIGN BENCHMARK (RED-BM)

RED-BM facilitates the analysis of reverse engineering approaches based on their ability to reconstruct class diagrams of legacy software systems. This is accomplished by offering the source code of projects of differing size and complexity as well as a number of reference UML models. The benchmark provides a set of measures that facilitate the comparison of reverse engineering results, for example class detection, to reference models including a *gold standard* and a number of meta-tools to aid in the analysis of tool outputs. The *gold standard* is a set of manually verified correct UML data, in whole or in part, for the artefacts.

The benchmark allows ranking of reverse engineering approaches by means of an overall performance measure that combines the performance of an approach with respect to a number of criteria, such as successful class or relationship detection. This overall measure is designed to be extensible through the addition of further individual measures to facilitate specific domains and problems. In addition, the benchmark provides analysis results and a ranking for a set of popular reverse engineering tools which can be used as a yardstick for new approaches. Full details, models, targets, results as well as a full description of the measurement processes used can be found at [7]. Although based on Java source code, the core concepts and measurements, such as detection of classes, relationships, and containers, are applicable to any object-oriented language and the benchmark could be extended to include other languages.

A. Target Artefacts

Our benchmark consists of a number of target software artefacts that originate from software packages of varying size and complexity. These include projects such as Eclipse, an open-source integrated development environment, and LibreOffice, a large popular open-source fully-featured office package, as well as some smaller custom examples.

Artefacts were chosen for inclusion on the basis that they provided a range of complexity in terms of lines of code and class counts, used a number of different frameworks, offered some pre-existing design information and were freely available for distribution (under an open-source licence). Two artefacts (ASCII Art Examples A and B) were created specifically for inclusion as a baseline offering a very simple starting point with full UML design and use of design patterns.

Cactus is also included as, although depreciated by the Apache Foundation, it has a number of existing UML diagrams and makes use of a wide number of Java frameworks. Eclipse was included primarily owing to a very large codebase which contains a varied use of techniques. The large codebase of Eclipse also provides for the creation of additional targets without incorporating new projects. JHotDraw has good UML documentation available both from the project itself and some third-party academic projects which sought to deconstruct it manually to UML. As with Eclipse, Libre Office provides a large set of code covering different frameworks and providing for more targets if required.

The benchmark artefact targets represent a range of complexity and architectural styles from standard Java source with simple through to high complexity targets using different paradigms, such as design patterns and presentation techniques. This enables a graduated validation of tools, as well as a progressive complexity for any new tools to test and assess their capabilities. The complete range of artefacts is shown in Table I, where large projects are broken down into constituent components. In addition, the table contains statistics on the number of classes, sub-classes, interfaces and lines of code for each of the artefacts. Also, included within RED-BM are a set of *gold standards* for class and relationship detection against which tool output is measured. These standards were created by manual analysis supported by tools, as described in Section IV.

TABLE I. SOFTWARE ARTEFACT TARGETS OF THE RED-BM

Software Target Artefact	**Main Classes**	**Sub Classes**	**Inter-faces**	**Lines of Code**
ASCII Art Example A				
Example A	7	0	0	119
ASCII Art Example B				
Example B	10	0	0	124
Eclipse				
org.eclipse.core. commands	48	1	29	3403
org.eclipse.ui.ide	33	2	6	3949
Jakarta Cactus				
org.apache.cactus	85	6	18	4563
JHotDraw				
org.jhotdraw.app	60	6	6	5119
org.jhotdraw.color	30	7	4	3267
org.jhotdraw.draw	174	51	27	19830
org.jhotdraw.geom	12	8	0	2802
org.jhotdraw.gui	81	29	8	8758
org.jhotdraw.io	3	2	0	1250
org.jhotdraw.xml	10	0	4	1155
Libre Office				
complex.writer	11	33	0	4251
org.openoffice.java. accessibility.logging	3	0	0	287
org.openoffice.java. accessibility	44	63	1	5749
All bundled code (sw + accessibility)	241	173	33	39896

B. Measuring Performance

RED-BM enables the systematic comparison and ranking of reverse engineering approaches by defining a set of *performance measures*. These measures differentiate the performance of reverse engineering approaches and are based on accepted quality measures, such as successful detection of classes and packages [8][9]. Although such functionality would be expected in reverse engineering tools, these measures provide a basic foundation for measurement to be built on, and represent the most common requirement in reverse engineering for detection of structural elements. Further, as seen in Section VI, these measures are alone capable of differentiating wide ranges of tool performance. The performance of tools with respect to a particular measure is expressed as the fraction of data that has been successfully captured. Therefore, these measures are built around determining the recall factor, e.g., how complete is the recovered set. Individual measures are then used in conjunction to form a weighted compound measure of overall performance. In our benchmark, we define three base measures to assess the performance of reverse engineering tools and approaches:

- **Cl:** The fraction of classes successfully detected
- **Sub:** The fraction of sub-packages successfully detected
- **Rel:** The fraction of relationships successfully detected (successful in that a relationship was detected and is of the correct type)

Each of these measures are functions that take a system to be reverse engineered s and a reverse engineering result r (i.e., a structural UML class diagram) that is produced by a reverse engineering approach when applied to s. The formal definition of our three base measures are as follows:

$$Cl(s,r) = \frac{C(r)}{C(s)} \ , \ Sub(s,r) = \frac{S(r)}{S(s)} \ , \ Rel(s,r) = \frac{R(r)}{R(s)} \quad (1)$$

where

$C(x)$ is the number of correct classes in *x*

$S(x)$ is the number of correct (sub-)packages in *x*

$R(x)$ is the number of correct relations in *x*

The overall performance *P* of a reverse engineering approach for the benchmark is a combination of these performance measures. The results of the measures are combined by means of a weighted sum, which allows users of the benchmark to adjust the relative importance of, e.g., class or relation identification. We define the overall performance of a reverse engineering approach that produces a reverse engineering result *r* for a system *s* as follows:

$$P(s,r) = \frac{w_{Cl}Cl(s,r) + w_{Sub}Sub(s,r) + w_{Rel}Rel}{w_{Cl} + w_{Sub} + w_{Rel}} \quad (2)$$

In this function, w_{Cl}, w_{Sub} and w_{Rel} are weightings that can be used to express the importance of the performance in detecting classes, (sub-)packages and relations, respectively. The benchmark results presented in this article all assume that these are of equal importance: $w_{Cl} = w_{Sub} = w_{Rel} = 1$, unless mentioned otherwise.

C. Complexity Categories

To further refine the evaluation of the reverse engineering capabilities of approaches we divide the artefacts of the benchmark into three categories of increasing complexity; C1, C2 and C3. These categories allow for a more granular analysis of tool performance at different levels of complexity. For example, a tool can be initially validated against the lowest complexity in an efficient manner only being validated against higher complexity artefacts at a later stage. Our complexity classes have the following boundaries:

- **C1:** $0 \leq$ number of classes ≤ 25
- **C2:** $26 \leq$ number of classes ≤ 200
- **C3:** $201 \leq$ number of classes

The complexity categories are based on the number of classes contained in the target artefact. As source code grows in size both in the lines of code and the number of classes it becomes inherently more complex and so more difficult to analyse [10], [11]. While a higher number of classes does not necessarily equate to a system that is harder to reverse engineer, we have chosen this metric as it provides a quantitative measure without subjective judgement.

The bounds for these categories were chosen as results demonstrated a noticeable drop-off in detection rates observed in the tools, as can be seen in Section VI. However, any user of the benchmark can introduce additional categories and relate additional performance measures to these categories to accommodate for large scale industrial software or more specific attributes such as design patterns. The extensibility aspect of our work is explained in more detail in Section III.

III. Extensibility of the Benchmark

A. Extensibility of Measurements

RED-BM's included performance measures provide a solid foundation to evaluate and compare current standards of reverse engineering. To accommodate the continual advancements in this field we have made the performance measure aspect of our benchmark extensible. Any user of the benchmark can introduce new performance measures, such as the fraction of successfully detected design patterns in a given code base. Once a *gold standard* has been determined for a specific detection within the artefacts it can be tested against tool output (as explained in Section II-C for the initial criteria). With these new measures the performance of approaches can be defined for specific reverse engineering areas. In a generalised fashion we define a performance measure to be a function *M* that maps a system *s* and its reverse engineering result *r* to the domain [0..1], where 0 means the worst and 1 the best possible performance.

In addition to providing means for creating new performance measures, we provide the possibility to create new compound performance measures (i.e., measures that are compiled from a set of individual performance measures). Formally, we define a compound measure to be a function *C* that maps a system *s* and its reverse engineering result *r* to the domain [0..1], where 0 means the worst and 1 the best possible performance:

$$C(s,r) = \frac{\sum_{i=1}^{n} w_i M_i(s,r)}{\sum_{i=1}^{n} w_i} \quad (3)$$

In this expression w_i is the weighting that determines the importance of the individual performance measure *i*. Note that the performance measures we introduced in Section II-B conform to this definition and, therefore, can be seen as an example of the extensibility of the benchmark.

To further illustrate how researchers and practitioners can use this mechanism to specialise the application of RED-BM we create a performance measure that acknowledges the capability of an approach to detect design patterns during reverse engineering. This is an active research field for which to the best of our knowledge a specialised benchmark is not available.

According to literature the detection of creational and structural design patterns is easier than behavioural design patterns [12]. Therefore, we introduce two new performance measures D_b for the successful identification of creational and structural design patterns(D_{cs}), and behavioural design patterns (D_b) for a system *s* and reverse engineering result *r*:

$$D_{cs}(p,s) = \frac{P_c(r) + P_s(r)}{P_c(s) + P_s(s)} \ , \ D_b(p,s) = \frac{P_b(r)}{P_b(s)} \quad (4)$$

where

$P_c(x)$ is the number of creational design patterns in *x*

TABLE II. Simplified Comparative XMI Output from Tools

ArgoUML	Enterprise Architect	Astah Professional
<**UML:Class** xmi.id = "…" name = "Circle" visibility = "package" …> <**UML:GeneralizableElement.generalization**> <**UML:Generalization** xmi.idref = "…" /> </**UML:GeneralizableElement.generalization**> … <**UML:Classifier.feature** <**UML:Operation** xmi.id = "…" name = "Circle" visibility = "public" …> </**UML:Class**>	<**packagedElement** xmi:id = "…" name = "Circle" visibility = "package" …> <**ownedOperation** xmi.id = "…" name = "Circle" visibility = "public" …> … <**generalization** xmi:type = "uml:Generalization" xmi:id = "…" general = "…" > </**packagedElement**>	<**UML:Class** xmi.id = "…" name = "Circle" version = "0" …> <**UML:ModelElement.namespace**> <**UML:Namespace** xmi.idred = "…" </**UML:Namespace** … <**UML:ModelElement.visibility** xmi.value = "package" /> … <**UML:GeneralizableElement.generalization**> xmi.idref = "…" /> xmi.idref = "…" /> </**UML:GeneralizableElement.generalization**>

$P_s(x)$ is the number of structural design patterns in *x*

$P_b(x)$ is the number of behavioural design patterns in *x*

In addition to these performance measures we introduce additional measures that demonstrate how to consider negative influences on performance. In this case, we consider falsely identified creational and structural design patterns (E_{cs}) and behavioural design patterns (E_b) a reverse engineering approach produces as part of the overall result:

$$E_{cs}(p,r) = 1 - \frac{F_c(r) + F_s(r)}{P_c(r) + P_s(r) + F_c(r) + F_s(r)} \quad (5)$$

$$E_b(p,r) = 1 - \frac{F_b(r)}{P_b(r) + F_b(r)} \quad (6)$$

where

$F_c(x)$ is the number of falsely identified creational design patterns in *x*

$F_s(x)$ is the number of falsely identified structural design patterns in *x*

$F_b(x)$ is the number of falsely identified behavioural design patterns in *x*

These individual performance measures for design patterns can now be combined into a single compound performance measure *DPR* for design pattern recognition in system *p* with reverse engineering result *r* that includes weightings for each individual component:

$$DPR(p,r) = \frac{w_{D_{cs}} D_{cs} + w_{D_b} D_b + w_{F_{cs}} F_{cs} + w_{F_b} F_b}{w_{D_{cs}} + w_{D_b} + w_{F_{cs}} + w_{F_b}} \quad (7)$$

B. Extensibility in Data Exchange

Another prominent aspect that needs to be addressed for a reusable and extensible benchmark is the gap that exists between input and output formats of various reverse engineering tools. Indeed, to make further use of reverse engineering output, for example, between tools or for re-projection of UML there is an Object Management Group (OMG) standard, the XML Metadata Interchange (XMI) format [13]. XMI is a highly customisable and extensible format with many different interpretations. Therefore, in practice, tools have a wide variation in their XMI output and exchange between reverse engineering tools, useful for interactive projection between tools without repetition of the reverse engineering process, is usually impossible. This variance in XMI format also hinders use of XMI data for further analysis outside of a reverse engineering tool, as individual tools are required for each XMI variation.

During the creation of the reverse engineering benchmark, two tools were developed, which could analyse Java source code identifying contained classes, and then, check for the presence of these classes within XMI output.

However, the need remained to make more generalised use of reverse engineering output XMI, beyond this specialist utility. Our research required the ability to load XMI into a memory model and manipulate and/or compare it. Additionally it was foreseen that future, as yet not specifically defined, uses could be found for programmatic access to reverse engineering output.

One of the challenges in making automated programmatic use of XMI output from different tools was the wide variety of output format. This is due to the wide range of customisation possibilities in the XMI format itself [13], it's parent Meta-Object Format (MOF; [14]), and the representation of UML elements within XMI [15]. Even the most basic structural elements such as classes, and relationships such as generalisation (inheritance) are represented in very different ways.

Such variation is shown in three XML listings showing partial example output for a class representation from ArgoUML, Enterprise Architect and Astah Professional (Table II).

Further work based upon the identification and analysis of variances within different reverse engineering tools' output, along with a desire to be able to integrate such output within more detailed analysis, led to the creation of a generic XMI parser (Section III-C). The parser solves the problem of XMI accessibility through generic use and abstract representation of structural data contained in XMI files of multiple formats. This parser is used by further tools for structural analysis or comparison as well as automated UML re-projection within Eclipse.

C. XMI Parser

The XMI Parser is a generic component designed for integration within other projects consisting of a Java package. The parser is capable of reading an XMI file, of most common output formats, recovering class and relationship information in a structured form. Data access classes are provided, which contain the loaded structural information, and can be accessed directly or recursively by third-party tools. As a self-contained utility package, the XMI Parser can be developed in isolation to tools making use of it and be incorporated into tools when required. A number of tools have been and continue to be developed within UEA to make use of reverse engineering information through implementation of the XMI Parser.

1) XMI Analyser: XMI Analyser uses the generic XMI Parser to load one or more XMI files which can then be analysed. Features include a GUI-based explorer showing the structure of the software and items linked through relationships. A batch mode can be used from the command line for automated loading of XMI files and analysis. XMI Analyser is primarily used for testing revisions to the XMI Parser, as an example application and also for the easy viewing of structural information contained within XMI, as shown in Figure 1.

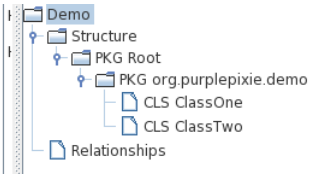


Figure 1. XMI Analyser Structure Display

XMI Analyser is also capable of comparison between multiple XMI files generating a report highlighting any differences found. This analysis can inform decisions as to the accuracy of the reverse engineering data represented in reverse engineering output.

2) Eclipse UMLet Integration: One of our desired outcomes was the ability to re-project UML outside of a specific reverse engineering tool. Such a capability would not only allow for detailed UML projections without access to the reverse engineering tool, but also programatic projection, for example in an interactive form. The Eclipse UMLet Integration, the interface of which is shown in Figure 2, is in the form of a plugin for the Eclipse Framework. The XMI Parser and supporting interfaces are included along with a graphical window-based interface and a visualisation component. This tool can load one or more XMI files and associate them with open or new UMLet documents. These documents can then be used to automatically generate a UML class diagram projection containing the structural elements contained within the XMI. An example of a re-projection within UMLet can be seen in Figure 3; please note, however, owing to a limitation in our UMLet API relationships are recovered but not shown.

3) Java Code Relation Analysis (jcRelationAnalysis): The *jcRelationAnalysis* tool is a generic utility designed to analyse and comprehend the relationship between elements (classes) in Java source code. This is accomplished by first building a structural picture of the inter-relationships between elements, such as classes, contained within a source code corpus, initially from reverse engineering output, for which the XMI Parser

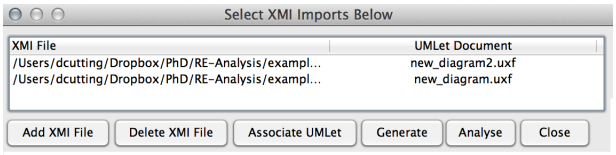


Figure 2. Eclipse Visualisation Interface

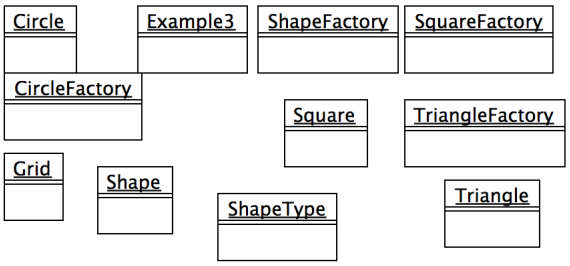


Figure 3. Eclipse UMLet Re-Projection of UML

is used. The ultimate intention of the tool is to work with combinational data from a number of different sources to compare or augment relationship information. This tool is now being used and further developed within our current and future research (Section IX).

IV. Benchmark Toolchain

The generic stages required to perform benchmarking are shown in Figure 4; the source code must be extracted from the project, the structural elements contained within the source code extracted directly and also by a reverse engineering tool, before the outputs are compared.

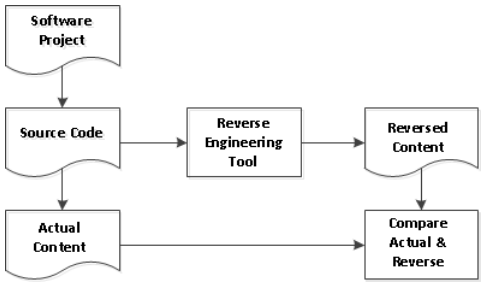


Figure 4. RED-BM Generic Process

To facilitate effective analysis and ease reproduction or repetition of the results a toolchain was developed for use within RED-BM, consisting of two main components (*jcAnalysis* and *xmiClassFinder*), combined to measure the rate of class detection. The steps followed in the application of the benchmark are shown in Figure 5 with the developed tools highlighted.

4) jcAnalysis: This tool recurses through a Java source tree analysing each file in turn to identify the package along with contained classes (primary and nested classes). The list of classes is then output in an intermediate XML format (DMI). For every target artefact, jcAnalysis' output was compared against a number of other source code analysis utilities, including within Eclipse, to verify the class counts. A manual

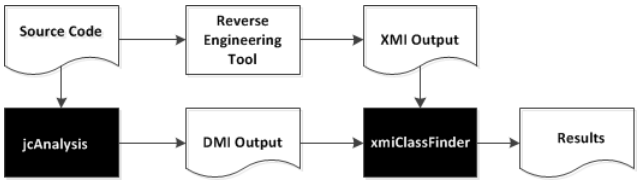


Figure 5. RED-BM Process with Toolchain Elements Highlighted

analysis was also performed on sections of source code to verify naming. Once verified, this output then constitutes the *gold standard* for class detection against which tool output is compared.

5) xmiClassFinder: This tool analyses an XMI file from a reverse engineering tool and attempts to simply identify all the classes contained within the XMI output (the classes detected by the reverse engineering tool in question). The classes contained within the XMI can be automatically compared to input from jcAnalysis (in DMI format) for performance (classes correctly detected) to be measured.

Once an analysis had been completed, a manual search was then performed on the source code, in XMI output, and within the reverse engineering tool itself, to try and locate classes determined as "missing" by the toolchain. This step also served to validate the toolchain, in that classes identified as "missing" were not then found to be actually present in the reverse engineering output.

V. Application of the Benchmark

To analyse the effectiveness of our benchmark, we have applied a range of commercial and open source reverse engineering tools (shown in Table III) to each target artefact. Each of the tools is used to analyse target source code, generate UML class diagram projections (if the tool supports such projections) and export standardised XMI data files. Although the source code target artefacts used for testing are broken down into the package level for analysis, the reverse engineering process is run on the full project source code to facilitate package identification. The output produced by each of the tools is subsequently analysed and compared to the generated *gold standard* using a benchmark toolchain we specifically created for comparison of class detection rates (see Section IV). Finally, we perform a manual consistency between the standard tool output and XMI produced to identify and correct any inconsistencies where a tool had detected an element but not represented it within the generated XMI. For this analysis we used weightings as stated, where all types of elements are of equal weight ($w_{Cl} = w_{Sub} = w_{Rel} = 1$), and categories of increased complexity have higher weight in the compound measure ($w_{C1} = 1, w_{C2} = 1.5, w_{C3} = 2$).

When analysing the results a wide range of variety was observed even for simple targets. Example A, one of the simplest targets with just 7 classes and two types of relationship, as depicted in Figure 6, demonstrates this variety. It can be seen in Figure 7 that Software Ideas Modeller failed to identify and display any relationship between classes. Other tools such as ArgoUML [16] (Figure 8) were very successful in reconstructing an accurate class diagram when compared to the original reference documentation.

TABLE III. List of Tools and Versions for Use in Evaluation

Tool Name (Name Used)	**Version Used (OS) Licence**
ArgoUML	0.34 (Linux) Freeware
Change Vision Astah Professional (Astah Professional)	6.6.4 (Linux) Commercial
BOUML	6.3 (Linux) Commercial
Sparx Systems Enterprise Architect (Enterprise Architect)	10.0 (Windows) Commercial
IBM Rational Rhapsody Developer for Java (Rational Rhapsody)	8.0 (Windows) Commercial
NoMagic Magicdraw UML (MagicDraw UML)	14.0.4 Beta (Windows) Commercial
Modeliosoft Modelio (Modelio)	2.2.1 (Windows) Commercial
Software Ideas Modeller	6.01.4845.43166 (Windows) Commercial
StarUML	5.0.2.1570 (Windows) Freeware
Umbrello UML Modeller (Umbrello)	2.3.4 (Linux) Freeware
Visual Paradigm for UML Professional (Visual Paradigm)	10.1 (Windows) Commercial
IBM Rational Rose Professional J Edition (Rational Rose)	7.0.0.0 (Windows) Commercial

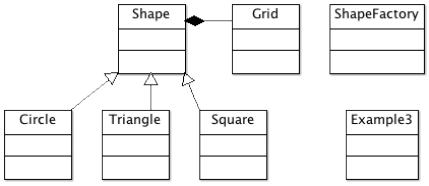


Figure 6. Reference Class Diagram Design for ASCII Art Example A

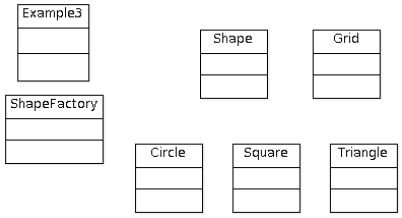


Figure 7. ASCII Art Example A Output for Software Ideas Modeller

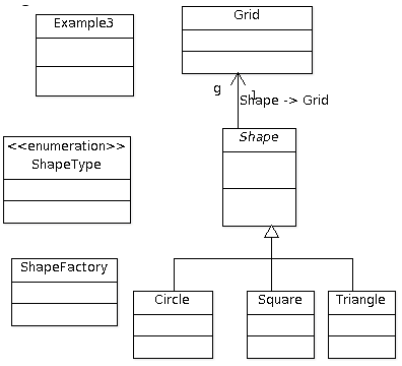


Figure 8. ASCII Art Example A Output for ArgoUML

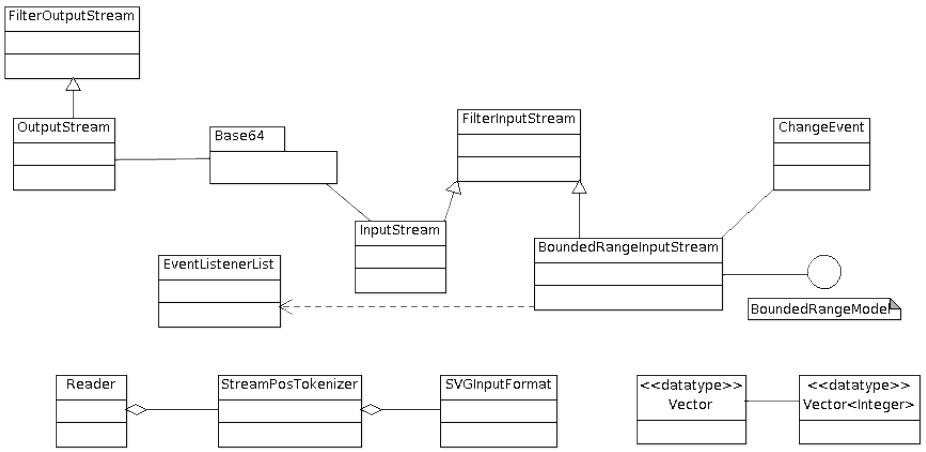


Figure 9. org.jhotdraw.io Output from Astah Professional (reconstructed)

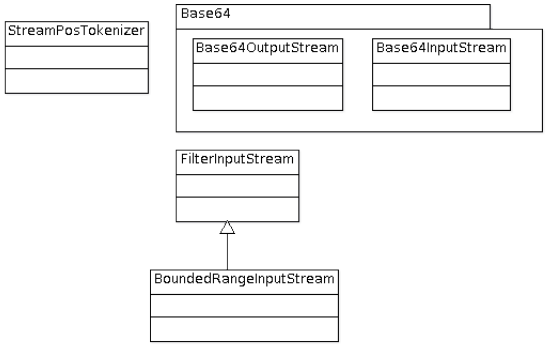


Figure 10. org.jhotdraw.io Output from Rational Rhapsody (reconstructed)

Figure 11. org.jhotdraw.io Output from ArgoUML

Another aspect in which difference is obvious relates to tool presentation, particularly when the target artefact is a Java package, which contains sub-packages nested to multiple levels. Some of the different ways tools visualise this, even for a single nesting level, is shown by the *org.jhotdraw.io* target. Tool output varies from a simple display of classes and packages at the top level (ArgoUML, Figure 11), a partial decomposition of top-level sub-packages showing contained constituent items (Rational Rhapsody, Figure 10), to a full deconstruction showing all constituent parts and relationships, but without indication of sub-package containment (Astah Professional, Figure 9).

In stark contrast to tools which performed well (e.g., Rational Rhapsody and ArgoUML) a number of tools failed to complete reverse engineering runs of benchmark artefacts and even crashed repeatedly during this procedure. The result of which is that they are classified as detecting 0 classes for those target artefacts. While some tools failed to output valid or complete XMI data, a hindrance to their usability and ease of analysis, this has not affected their performance evaluation as their performance could be based on our manual analysis of their UML projection.

TABLE IV. CRITERIA RESULTS BY TOOL

Criterion > ∨ **Tool**	**CD** %	**C1** %	**C2** %	**C3** %	**CM** %
ArgoUML	100	98.15	75	100	88.27
Astah Professional	100	97.62	100	100	99.47
BOUML	100	92.59	75	100	86.42
Enterprise Architect	100	66.67	62.22	100	80.00
Rational Rhapsody	100	100	100	100	100.00
MagicDraw UML	100	98.15	100	100	99.38
Modelio	47.33	95.92	29.66	12.02	36.54
Software Ideas Modeller	86.41	62.15	41.48	46.04	48.10
StarUML	47.11	47.22	23.47	31.16	32.17
Umbrello	9.2	35.79	5.95	0	9.94
Visual Paradigm	12.42	38.18	51.68	16.67	33.12
Rational Rose	8.69	38.05	1.09	0	8.82

VI. EVALUATION OF ANALYSIS RESULTS

For the analysis of the results produced by the reverse engineering tools, we use a standard class detection performance measure for all targets (*CD*, formula (1)). Artefact results are broken into complexity categories as defined in Section II-C.

Finally, we use the compound measure *CM* (as defined in Section II-B, formula (3)), which contains the three complexity measures with weighting as follows: $w_{C1} = 1, w_{C2} = 1.5, w_{C3} = 2$; giving a higher weighting to target artefacts that contain more lines of code.

Using these performance measures a wide range of results between the tools used for analysis can be seen. Some tools offer extremely poor performance, such as Rational Rose and Umbrello, as they crashed or reported errors during reverse engineering or UML projection, failing to detect or display classes and relationships entirely for some targets. As a general trend, the percentage of classes detected on average declined as the size of the project source code increased. As the number of classes detected varied significantly in different tools (Figure 12) so did the amount of detected relationships, to a degree this can be expected as if a tool fails to find classes it would also fail to find relationships between these missing classes. In this figure, the difference between the standard class detection measure CD and the compound measure CM becomes clear

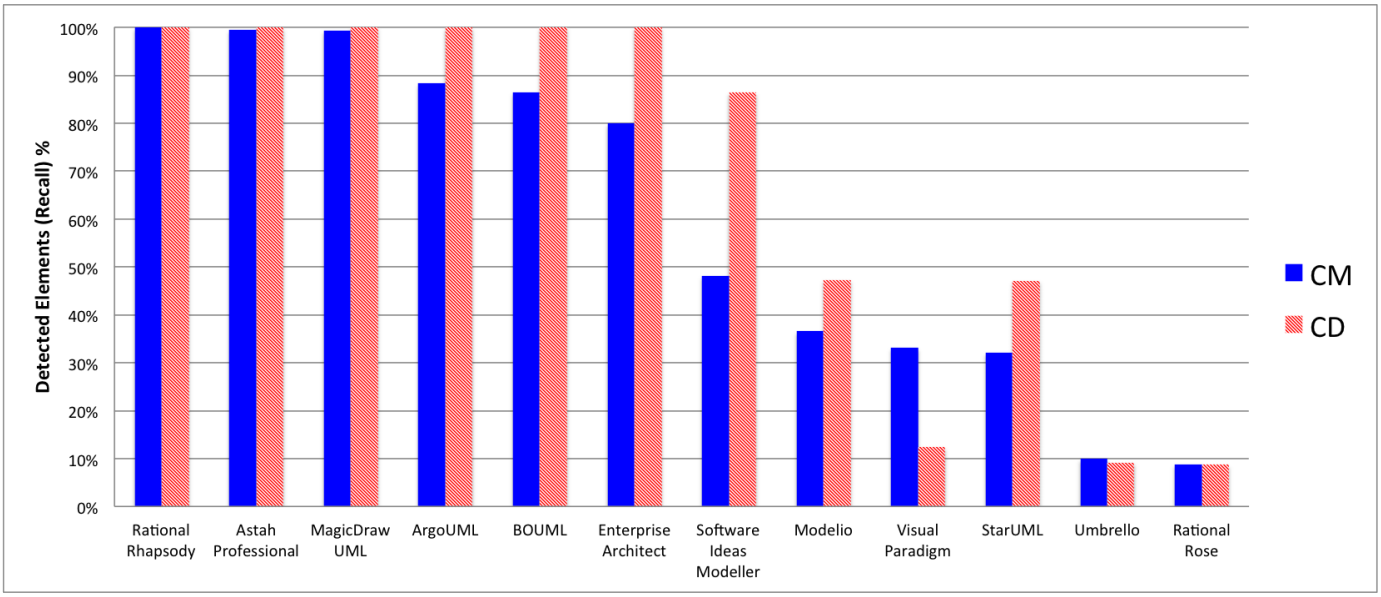


Figure 12. Overall Class Detection (CD) and Compound Measure (CM) Performance by Tool

as, for example, ArgoUML was very strong in class detection but performed at a slightly lower level on relation detection, which is explicitly considered in the compound measure. It is also interesting to note that Visual Paradigm offered better performance for the compound measure as opposed to class detection highlighting its superior ability to deal with relations and packages as compared to class detection.

Overall, our benchmark identified IBM Rational Rhapsody as the best performer as it achieved the maximum score for our compound measure (100%) with two other tools, Astah Professional and MagicDraw UML coming in a close second scoring in excess of 99%. As the poorest performers our work highlighted Umbrello, Visual Paradigm and notably IBM Rational Rose which scored the lowest with a compound measure of just 8.82% having only detected 8.69% of classes. A detailed breakdown of the performance of the tools for individual targets is provided with the benchmark [7].

This range of performance scores clearly shows a very marked differentiation between tools. At the top end some six tools score 80% or above in the compound measure, with three over 90%. In most a clear drop-off in detection rates are seen in the complexity measures as the size and complexity of the targets increase with an average measure score of 73.47%, 58.70% and 54.66% through the complexity categories C1, C2 and C3, respectively (Table IV and Figure 13).

There is a noticeable distribution of tool performance for the compound measure; five score under 40%, six score in excess of 80% and only one lies in the middle (48.1%).

It is interesting to note that of the top four performing tools three are commercial with ArgoUML, a freeware tool, scoring 88.27%. This makes ArgoUML a significantly better performer than well-known commercial solutions such Software Ideas Modeller and Rational Rose. For complete results, targets and reference documentation for this analysis please visit the benchmark website [7].

Although outside the scope of this paper, in general we found that the workflow processes of some tools were much more straightforward than others. For example, Change Vision Astah Professional and IBM Rational Rhapsody provided for straightforward generation of diagrams with configurable detail (such as optional inclusion of members and properties within class diagrams) either during or immediately after reverse engineering. On the other hand, tools such as BOUML and IBM Rational Rose required considerable manual effort in the generation of class diagrams with the need for individual classes to be placed in the diagram although relationships between classes were automatically generated. For a number of tools the lack of usability was further aggravated as their reverse engineering process repeatedly crashed or returned numerous errors on perfectly valid and compilable source code.

VII. Discussion

In the previous sections, we have demonstrated the ability of RED-BM to assess and rank reverse engineering approaches. In the following, we discuss the accuracy and validity of our approach. The targets in RED-BM currently range in size from approximately 100 to 40,000 lines of code. It can be argued that this is not representative of industrial software systems that can consist of millions of lines of code. In practice, however, reverse engineering activities tend to focus on specific areas of limited size rather than reconstructing the entire design of a very large system in a single pass [3] making our targets representative for a typical application scenario. This is supported by the capability of our benchmark to provide targets diverse enough to be able to classify the performance of a range of industry standard tools.

RED-BM currently supports the evaluation of reverse engineering approaches that focus on traditional design elements such as classes, packages and relations. It is possible that novel reverse engineering approaches will be introduced that focus on more complex design elements such as design patterns, traceability links, etc., and are beyond the current evaluation capabilities of RED-BM. However, the evaluation capabilities of the benchmark can be improved by using the extension

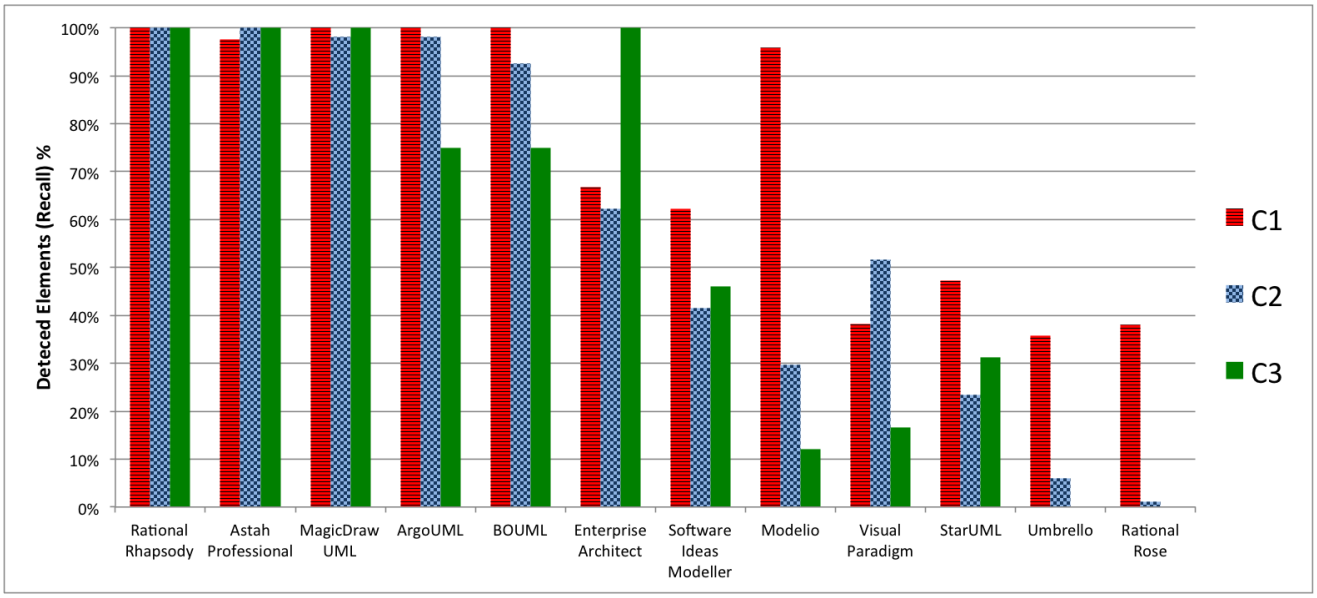


Figure 13. Tool Performance by Complexity Criteria

mechanism illustrated in Section III. Using this mechanism new performance criteria and measures can be defined that explicitly take more advanced properties of reverse engineering approaches into account.

A final point we would like to highlight is that the effort involved in evaluating tools and approaches requires the ability to efficiently compare their outputs. While there is a standardised output format available in the shape of XML Metadata Interchange (XMI) files, a wide variety of implementations exist which makes its use impractical. This can severely inhibit the application of the benchmark. To accommodate this we provide a number of utilities which can be used to a) analyse Java source code, b) analyse XMI output and c) identify components missing from the XMI output, which were found in the source code. Use of these utilities drastically reduces the time required to compare tool performance in terms of class, package and relation detection.

VIII. Related Work

The use of benchmarks as a means to provide a standardised base for empirical comparison is not new and the technique is used widely in general science and in computer science specifically. Recent examples where benchmarks have been successfully used to provide meaningful and repeatable standards include comparison of function call overheads between programming languages [17], mathematical 3D performance between Java and C++ [18], and embedded file systems [19]. Our benchmark provides the ability for such meaningful and repeatable standard comparisons in the area of reverse engineering.

These previous benchmarks and others which have been reviewed (such as [20], [21], [5], and [22]) share many common features in their structure and presentation which have been incorporated into this benchmark and paper. Such is the perceived importance of benchmarks to support empirical comparison that the Standard Performance Evaluation Corporation are in the process of forming sets of standards to which benchmarks in certain areas of computer science can be created to [23].

Previous work reviewing reverse engineering tools has primarily focused on research tools many with the specific goal of identification of design patterns [3], [4], [24], [12], [25], clone detection [26] or a particular scientific aspect of reverse engineering such as pattern-based recognition of software constructs [27]. A previous benchmarking approach for software reverse engineering focused on pattern detection with arbitary subjective judgements of performance provided by users [5]. The need for benchmarks within the domain of reverse engineering to help mature the discipline is also accepted [6].

This previous work defines the importance of reverse engineering in industry as well as a research challenge. Our benchmark is a novel yet complimentary approach to previous reverse engineering benchmarks, providing a wider set of target artefacts and tool analysis than those just focused on design patterns or other specific outputs. As such it provides a solid framework for the generalised comparison of reverse engineering tools with the option of extensibility when specific measurements are required and also allows for integrating previous efforts into a single benchmark.

IX. Conclusion and Future Direction

In this paper we introduced RED BM, a benchmark for evaluating and comparing reverse engineering approaches in a uniform and reproducible manner. To analyse the effectiveness of RED-BM we applied it to a range of reverse engineering tools, ranging from open source to comprehensive industrial tool suites. We demonstrated that RED-BM offers complexity and depth as it identified clear differences between tool performance. In particular, using the compound measure (CM) RED-BM was capable of distinguishing and ranking tools from very low (8.82%) to perfect (100%) performance. The benchmark is inherently extensible through the definition of

further measures, changes to weighting to shift focus, and the creation of new compound measures.

The XMI Parser allows tools to make direct use of reverse engineering output overcoming the fragmentation issues. The capability of direct use of reverse engineering output is clearly demonstrated through the ability for UML to be re-projected within UMLet, and also used in other tools for further analysis.

The future direction of our work will be to combine reverse engineering output with other sources of information about a source corpus, for example mining repository metadata or requirement documentation. The *jcRelationAnalysis* tool is being used as a programmable basis for integration of different sources of information into a common format of relationships between source code elements. These relationships, be they direct and found through reverse engineering, such as generalisations, or semantic in nature and found through other means, will be used in combination to form a more complete understanding of a software project.

Such analysis will aid both general comprehension of software and also change impact analysis by identifying relationships between elements not immediately obvious at the code or UML level.

REFERENCES

[1] D. Cutting and J. Noppen, "Working with reverse engineering output for benchmarking and further use," in Proceedings of the 9th International Conference on Software Engineering Advances. IARIA, Oct. 2014, pp. 584–590. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=icsea_2014_21_40_10425

[2] G. Rasool and D. Streitfdert, "A survey on design pattern recovery techniques," International Journal of Computing Science Issues, vol. 8, 2011, pp. 251–260.

[3] J. Roscoe, "Looking forwards to going backwards: An assessment of current reverse engineering," Current Issues in Software Engineering, 2011, pp. 1–13.

[4] F. Arcelli, S. Masiero, C. Raibulet, and F. Tisato, "A comparison of reverse engineering tools based on design pattern decomposition," in Proceedings of the 2005 Australian Software Engineering Conference. IEEE, 2005, pp. 262–269.

[5] L. Fulop, P. Hegedus, R. Ferenc, and T. Gyimóthy, "Towards a benchmark for evaluating reverse engineering tools," in Reverse Engineering, 2008. WCRE'08. 15th Working Conference on. IEEE, 2008, pp. 335–336.

[6] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using benchmarking to advance research: A challenge to software engineering," in Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, 2003, pp. 74–83.

[7] UEA, "Reverse engineering to design benchmark," http://www.uea.ac.uk/computing/machine-learning/traceability-forensics/reverse-engineering, 2013, [Online; accessed May 2013].

[8] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," Journal of Software Maintenance and Evolution: Research and Practice, vol. 15, no. 2, 2003, pp. 87–109.

[9] G.-C. Roman and K. C. Cox, "A taxonomy of program visualization systems," Computer, vol. 26, no. 12, 1993, pp. 11–24.

[10] N. E. Fenton and S. L. Pfleeger, Software metrics: a rigorous and practical approach. PWS Publishing Co., 1998.

[11] B. Bellay and H. Gall, "A comparison of four reverse engineering tools," in Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on. IEEE, 1997, pp. 2–11.

[12] I. Philippow, D. Streitferdt, M. Riebisch, and S. Naumann, "An approach for reverse engineering of design patterns," Software and Systems Modeling, vol. 4, no. 1, 2005, pp. 55–70.

[13] OMG et al., "OMG MOF 2 XMI Mapping Specification," http://www.omg.org/spec/XMI/2.4.1, 2011, [Online; accessed December 2012].

[14] OMG, "OMG Meta Object Facility (MOF) Core Specification," http://www.omg.org/spec/MOF/2.4.1, 2011, [Online; accessed December 2012].

[15] OMG, "Unified modelling language infrastructure specification," http://www.omg.org/spec/UML/2.0/, 2005, [Online; accessed December 2012].

[16] ArgoUML, "Argouml," http://argouml.tigris.org/, 2012, [Online; accessed December 2012].

[17] A. Gaul, "Function call overhead benchmarks with matlab, octave, python, cython and c," arXiv preprint arXiv:1202.2736, 2012.

[18] L. Gherardi, D. Brugali, and D. Comotti, "A java vs. c++ performance evaluation: a 3d modeling benchmark," Simulation, Modeling, and Programming for Autonomous Robots, 2012, pp. 161–172.

[19] P. Olivier, J. Boukhobza, and E. Senn, "On benchmarking embedded linux flash file systems," arXiv preprint arXiv:1208.6391, 2012.

[20] Q. Quan and K.-Y. Cai, "Additive-state-decomposition-based tracking control for tora benchmark," arXiv preprint arXiv:1211.6827, 2012.

[21] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf, "Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results," Annals of Operations Research, vol. 194, no. 1, 2012, pp. 59–70.

[22] A. Klein, A. Riazanov, M. M. Hindle, and C. J. Baker, "Benchmarking infrastructure for mutation text mining," J. Biomedical Semantics, vol. 5, 2014, pp. 11–24.

[23] W. Bays and K.-D. Lange, "Spec: driving better benchmarks," in Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering. ACM, 2012, pp. 249–250.

[24] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo, "Design pattern detection using software metrics and machine learning," in First International Workshop on Model-Driven Software Migration (MDSM 2011), 2011, p. 38.

[25] N. Pettersson, W. Lowe, and J. Nivre, "Evaluation of accuracy in design pattern occurrence detection," Software Engineering, IEEE Transactions on, vol. 36, no. 4, 2010, pp. 575–590.

[26] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," Software Engineering, IEEE Transactions on, vol. 33, no. 9, 2007, pp. 577–591.

[27] M. Meyer, "Pattern-based reengineering of software systems," in Reverse Engineering, 2006. WCRE'06. 13th Working Conference on. IEEE, 2006, pp. 305–306.

Towards a Classification Schema for Development Technologies: an Empirical Study in the Avionic Domain

Davide Taibi, Valentina Lenarduzzi
Free University of Bolzano-Bozen
Bolzano-Bozen, Italy
{davide.taibi, valentina.lenarduzzi}@unibz.it

Laurent Dieudonné
Liebherr-Aerospace
Lindenberg, Germany
laurent.dieudonne@liebherr.com

Christiane Plociennik
University of Kaiserslautern
Kaiserslautern, Germany
christiane.plociennik@cs.uni-kl.de

Abstract— **Software and hardware development organizations that consider the adoption of new methods, techniques, or tools often face several challenges, namely to: guarantee process quality, reproducibility, and standard compliance. They need to compare existing solutions on the market, and they need to select technologies that are most appropriate for each process phase, taking into account the specific context requirements. Unfortunately, this kind of information is usually not easily accessible; it is incomplete, scattered, and hard to compare. Our goal is to report on an empirical study with high-level practitioners, to extend our previous work on a classification schema for development technologies in the avionic domain. We investigate the acceptance and the possible improvements on the schema, with the aim to help decision makers to easily find, compare and combine existing methods, techniques, and tools based on previous experience. The study has been carried out with five technical leaders for the development of flight control systems, from Liebherr-Aerospace Lindenberg GmbH and the results show that the schema helps to transfer knowledge between projects, guaranteeing quality, reproducibility, and standard compliance.**

Keywords-component; process improvement; technology classification; technology selection; tool selection; method selection; process configuration.

I. INTRODUCTION

The definition of a product development process that guarantees quality and reproducibility often takes years. Moreover, in certain domains, such as avionics, the process must comply with a set of standards.

The introduction of a new technology may break the consistency and standards compliance of the process. To limit this risk, two major aspects must be considered. First, the objectives and prerequisites for each process step must be fully documented and structured. Second, the contribution of each method and tool intended to be used, must be limited to the objectives set by each domain process activity and their role in each process step must be fully described.

A structuring framework, enabling the classification of the technologies in process activities would speed up the integration of new technologies and contribute to guaranteeing compliance with the company processes.

To facilitate the classification of technologies, the Reference Technology Platform (RTP) has been developed. RTP is a set and arrangement of methods, workflows, and tools that allow interaction and integration on various levels in order to enable efficient design and development of (complex) systems [1] [3].

In the context of the ARAMiS project (Automotive, Railways, Avionics Multicore Systems) [4], a classification schema based on the RTP has been developed. It classifies technologies along two dimensions: abstraction levels and viewpoints. In our previous work, we introduced how RTP and Process Configuration Framework (PCF) could have been applied in the avionic domain [1].

The goal of this paper is to conduct an empirical study with the goal of evaluating the RTP and PCF approaches, for the purpose of understanding their acceptance and applicability to the selection process, in the context of new product development in the avionic domain.

For this purpose, in this paper, we present the results of the case study proposed in [1] and we conduct an empirical study so as to validate the approach with high-level practitioners from Liebherr-Aerospace.

The results of this work suggest that the classification provides a useful framework for decision makers and allows them to base their decisions on previous experience instead of on personal opinions. Moreover, the classification allows them to guarantee process quality, reproducibility and standards compliance, facilitating knowledge transfer from project to project or between employees.

The remainder of this paper is structured as follows: Section II describes related work; Section III introduces the classification schema and its implementation in PCF, while Section IV describes the avionic use case. In Section V, we describe the Empirical Study while in Section VI we report results of the study. Finally, we draw conclusions in Section VII and provide an outlook on future work.

II. RELATED WORK

Here, we present some common technology classification schemas.

In 1987, Firth et al. published an early classification schema [5]. In this work, software development methods are

classified according to two dimensions: the stages of the development process (specification, design, and implementation) and the view (functional, structural, and behavioral). The stages are specification, design, and implementation; the views are functional, structural, and behavioral. Our schema, too, is two-dimensional, and our viewpoints dimension is similar to the views dimension of Firth et al. However, Firth et al.'s second dimension is concerned with the process stages, which we map onto the viewpoints dimension. Instead, the second dimension in our schema is concerned with abstraction levels.

In the late 1980's, the idea of the Experience Factory was first published [6]. It was then updated in 1991 [7] and in 1994 [8]. The idea is to describe software development artifacts in so-called experience packages and to include empirical evidence on how these artifacts have been used previously. This way, the Experience Factory provides a comprehensive framework for the reuse of software. The goal is to enable software engineers to base their decisions on company experience.

The Experience Factory is a more general concept than ours. In the Experience Factory, any software engineering artifact can be an object for reuse, e.g., products or requirements documents. Moreover, a specific schema for storing different technologies for reuse is not provided in the Experience Factory. Neither does it include any algorithms to search for or to combine technologies.

The C4 Software Technology Reference Guide (C4 STR) is a catalog that contains more than 60 technologies. It constitutes an alternative approach to technology classification and was developed in parallel to the later versions of the Experience Factory.

In comparison to our work, the C4 STR schema includes a large number of technologies. However, the attributes it uses are not as detailed as those in our schema, and it includes no reference to context or to impact.

Later, Birk merged the Experience Factory approaches with the C4 STR [8]. This evolved into the concept of experience management in the late 1990's. This work served as the basis for other publications that evolved this schema and extended the idea of the Experience Factory [9].

A classification schema for software design projects was developed by Ploskonos [10]. With the help of this schema, generic process descriptions and methods can be adapted to individual processes more easily. It classifies design projects into one of the four groups Usability, Capability, Extension, and Innovation. Each of these groups is associated with specific process characteristics in order to help the user in setting up the actual process. Ploskonos' approach is more narrow than ours: It classifies processes with respect to the project type, ignoring other characteristics, e.g., project size or domain.

III. THE CLASSIFICATION SCHEMA

As a foundation for the case study, which we present in the next section, we now introduce the classification schema we applied. The goal of the schema is to provide a complete engineering tool chain that can be used to collect and integrate technologies. This way, the schema supports the activities required for a structured development process.

With our schema, we address the development of industrial projects that are big and complex. Typically, such projects run several years and require the joint efforts of many employees.

In the industry, requirements-based process models are commonly used to plan the different baselines and to ensure that these baselines are accomplished on time in different phases of realization. Usually, every phase and every step of the processes produces artifacts which then constitute the inputs for the next phase(s) or step(s). These process models are based on, or are extensions of, the V-Model [11].

An instance of the V-Model for the avionics domain is shown in Figure 1. It is an extract of the avionics standard SAE ARP4754A [12], and it includes the interaction between both avionics development and safety integral processes.

Traditionally, the V-Model is used in the iterations that are carried out in order to accomplish each baseline. In addition to the iterations, concepts such as the definition of phases, the definition of objectives, periodical assessments, the definition of roles, and traceability (forward and backward) are traditionally included in these development processes. Current agile methodologies, like SCRUM [11], have also been inspired by these concepts.

The schema we present in this paper serves as a generic development model that covers the industrial development processes. Naturally, the instances of this generic development model depend both on the development standards in the industry and on the particular company.

Using the information provided in the schema, decision makers can find the most appropriate technologies based on the technologies' interaction and integration on multiple levels. This helps to efficiently design and develop complex systems. Moreover, the schema may provide an overview of the tools and methods used in previous projects. As the activities inherent to the industrial processes (e.g., planning phases, assessment meetings and accomplishment summaries) are performed periodically throughout each project development, a huge amount of data can be collected during the development life cycle of every project. This data includes, for example, the decisions made, or the quality and special uses of the tools, technologies and methods. This helps to build a knowledge base that is adapted to the company's development processes and addresses best practices as well as pitfalls. Thus, new projects can benefit from prior experience instead of starting from scratch.

Furthermore, the schema can help new employees to quickly become familiar with the tools and methods available in the company for every phase of the development process fostering knowledge transfer within a company.

Inspired by the work done in SPES2020 [13] and SPES_XT [14], our schema can be envisaged as a two-dimensional matrix, where viewpoints form the columns and abstraction levels form the rows. The viewpoints dimension consists of “Requirements”, “Functional”, “Logical”, and “Technical”.

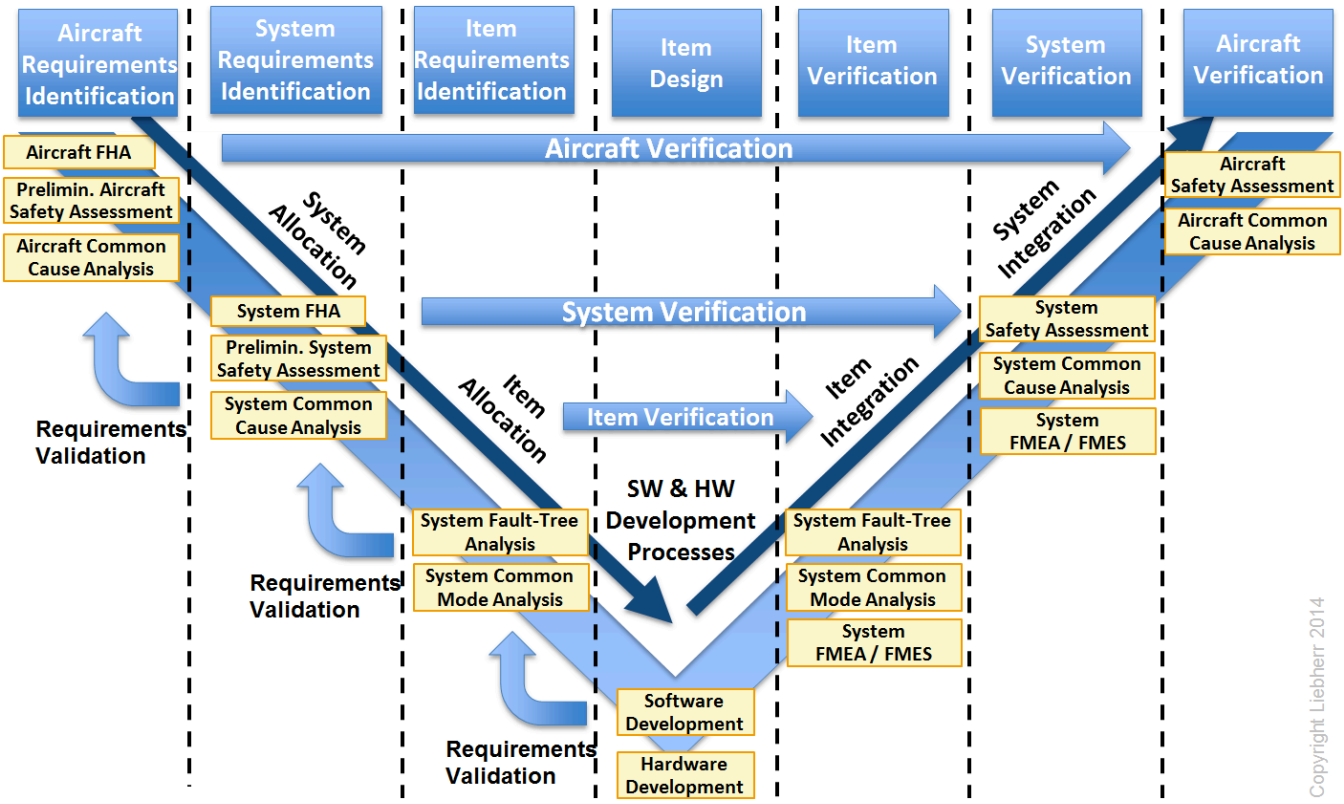


Figure 1: Avionics V-Model extract from the ARP4754A [12]

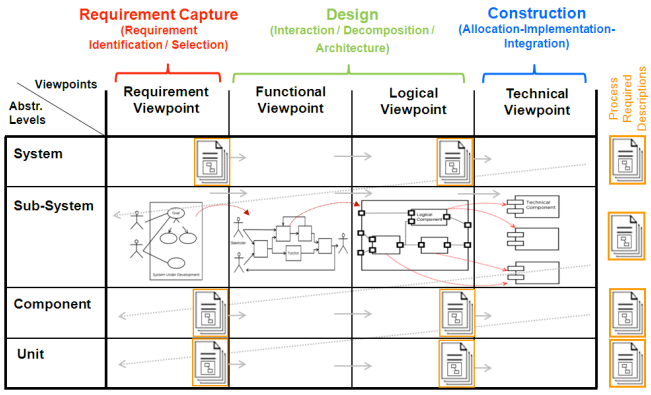


Figure 2: Generic representation of our classification schema

Those four viewpoints can be mapped to the three phases of the development process: the requirements viewpoint corresponds to the requirements capture phase, the functional and logical viewpoints can be mapped onto the design phase, and the technical viewpoint corresponds to the construction or implementation phase (see Figure 2). Figure 2 depicts the generic version of the schema. The abstraction levels correspond to different decompositions of the system. These are (from coarse-grained to fine-grained): system, sub-systems, components, and units. This generic set of abstraction levels can be substituted by different, domain-specific abstraction levels according to the specific application domain (automotive, railways, avionics etc.). For instance, the avionics domain defines the following abstraction levels (see Figure 3): "Aircraft", "System", "Equipment", and "Item".

Each step of the product development process that must be carried out is represented as a cell in the schema, to be traversed from the top left cell to the rightmost. This is represented by the arrows in Figure 2 and Figure 3.

Each step produces artifacts as outputs. These outputs may contribute directly to the accomplishment of the process objectives required by the domain, or indirectly if they serve as inputs for other cells in later steps. The objectives specified by the domain process depend on the development phase and the abstraction level.

Here, we explain how the matrix is traversed, as shown in Figure 3. We start at a given abstraction level. First, the requirements related to this abstraction level are recorded in the requirement viewpoint. The outputs of this viewpoint are the filtered requirements, applicable for the (sub…)system under focus. They are needed in order to start the design of the (sub…)system. The design phase comprises the functional and the logical viewpoint. In the functional viewpoint, the network of functions representing the system workflow is determined. It is then undertaken into the logical viewpoint, where a structuration (decomposition and/or composition) of the identified functions is performed. If the objectives of the logical viewpoints are fulfilled, we move on to the technical viewpoint. Here, the construction of the system is started.

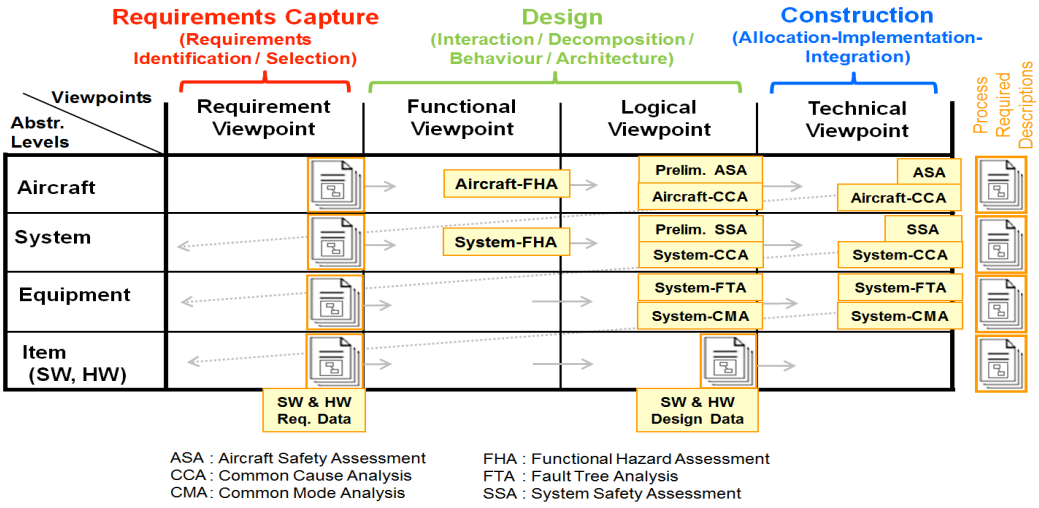


Figure 3: Example of classification schema for the avionics domain.

Sometimes iterations must be carried out, e.g., to introduce new requirements or to incorporate realization constraints that appear a posteriori and that influence the design of the system.

If not all requirements derived from the design (and hence from the requirement viewpoint) have been fulfilled at the end of the abstraction level, the unfulfilled requirements are used as a basis in the next abstraction level. They are recorded in the requirement viewpoint of this new current abstraction level. Now the steps described above for the previous abstraction level are carried out again for the next abstraction level.

In order to foster partial and iterative development, a set of transition criteria is defined. These transition criteria control the transition from one cell to the next. With the help of transition criteria, it is possible to evaluate the risks of commencing the next development step if not all objectives of the current step are fulfilled. It is then possible to control the current status of fulfillment of the objectives, which will be realized after several iterations.

In order to fulfill the objectives of each step, the methods used by the system and software engineers are usually supported by tools. Which methods and tools are required depends on the specific characteristics of the respective development process: the category of product that is to be developed, the requirements, the abstraction level, and the focus of the current development iteration (e.g., the objectives to be addressed). Furthermore, the integration of the technology chain used may also differ. The methods must as well support the transition criteria between the process steps.

A. The implementation of the classification schema in PCF

The proposed schema has been implemented as a web application in the PCF tool [15]. PCF is an online platform, developed by means of the Moonlight SCRUM process [16][17]. PCF allows users to search for technologies based on abstraction levels and viewpoints as defined in the schema. Furthermore, PCF adds two more aspects to provide information about previous experience using a specific technology: Context and Impact. Hence, the data schema in PCF is based on three models as defined in [18] (as shown in Figure 6):

- *Technology*: includes a set of attributes for describing a technology in as much detail as possible.
- *Context*: includes information on the context, such as application domain, project characteristics, and environment in which the respective technology has been applied.
- *Impact:* includes previous experience on applying a specific technology in a specific context.

The PCF tool contains a search feature that allows users to search for technologies based on the attributes defined in the models in Figure 6. This enables the user to search for technologies used in projects with specific characteristics, e.g., projects fulfilling a certain industrial standard.

Basic use cases for PCF, as shown in Figure 4, are:

- Search for a technology based on context requirements (not mandatory)
  - List view
  - Matrix view
- View details for a technology
- View related context
- View details for a context
- View related impacts
- View details for a related impact

Moreover, PCF implements the schema for different domains (avionics, automotive, and railways).

Figure 5 shows an example of the schema represented in PCF for the avionics domain. This figure includes the methods mentioned in the use case or directly the tools realizing them, as well as several other technologies for the avionics domain in addition to those mentioned above. In this version of the tool, we do not consider interoperability issues. The next version of the tool will address the challenge of interoperable tool chains.

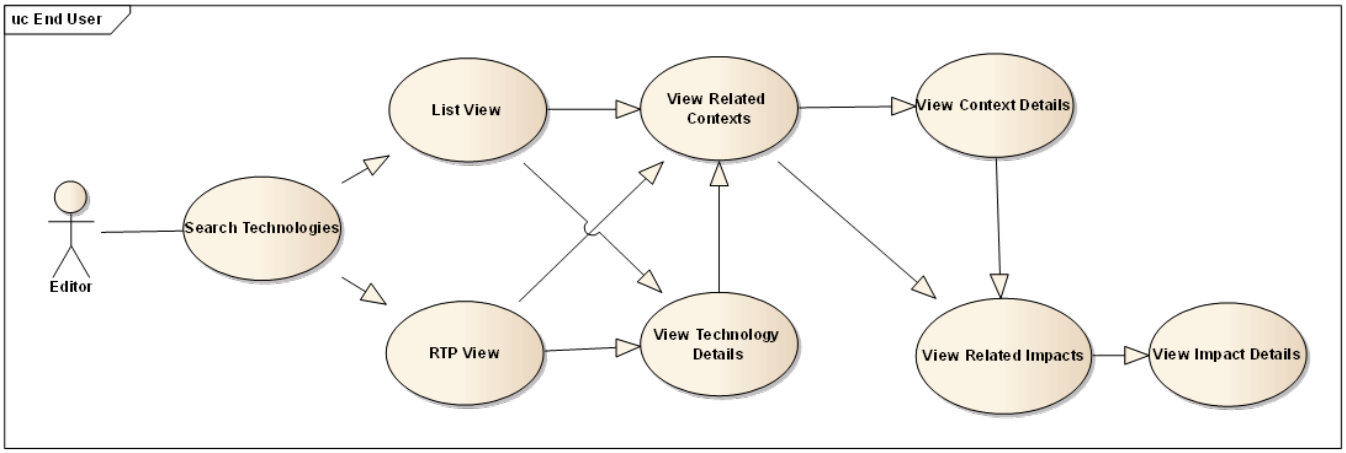


Figure 4: PCF Use Cases

Traditionally, at the aircraft and system abstraction level, but also partly persistent at the lower levels, mainly few and text-based tools (IBM Rational DOORS, MS-Excel, MS-Word, …) are used, completed pointwise with advanced graphical tools (MS-Visio, etc.) for architecture overview, and with specific tools simplifying the validation and verification of the system under development. Model-based methods and tools appears more and more for parts of the functional aspects needing to be simulated, or where better structuration, formalization and automation can be obviously performed to save time and money (SysML/UML technologies, MATLAB/Simulink, ESTEREL Scade, etc.). The model-based development methods facilitate an overview of the system, but need a strong defined formalism to be uniquely understandable. Structured text can be more precise with less formalism, but for big projects, many additional informative descriptions or pictures are needed to keep the red line with acceptable workload, in particular for engineers having to work with these requirements for the next development step. A mix of the both methods is probably the most efficient, if interoperability between the tools is provided. Both are also accurate enough to ensure exact traceability with a minimum of orderliness.

Thanks to the structured methodology, to the overview and to the collection of experience enabled by the PCF, development tools offering more automatisms but also being complex to integrate, can be easier incorporated in development processes. An example of possible enhanced tool environment is done in Figure 5. Some details about inputs-outputs are given in section IV, but it is not the goal of this paper to describe the details of use of each tool – this is also depending of company processes. Attributes to evaluate the quality, the adequacy and the added value of the tool are integrated in the PCF template by filling the technology, context and impact information like defined in Figure 6. For example, a tool having a qualification kit for the automation of a specific process step (e.g., code generation with ESTEREL Scade) provides a substantial advantage by avoiding manual work like a review activity, which saves much development time.

But its integration in the development process has also an impact on recurring and non-recurring costs, among other concerning purchase, training or maintenance fee. At the end, a trade-off decision must be taken to select the adequate chain of technologies and tools which could support an optimal project budget.

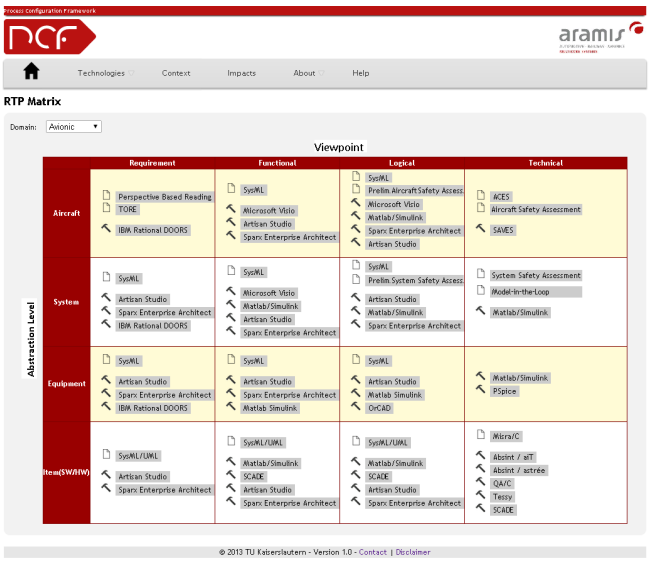


Figure 5: An example of the schema in the avionic domain implemented in PCF.

IV. Applying The Classification Schema in The Avionic Domain

In this section, we sketch an example of a use case of the classification schema in the avionics domain.

In the avionic industry, two main processes are defined and address two different aspects corresponding to the two branches of the V-Model: the Development Process and the Integral Process [12] (see Figure 1). The combination of both main processes defines abstraction levels (Aircraft, System, Equipment/Item, Software, Hardware, etc.) and specific processes for each of them. Iterations can be done inside an abstraction level, or inclosing them.

Technology

Element	Attributes
Name	Name
	Abbreviation
Description	Short description
	Long description
	SPEM model
	Belongs to family
	Complements
	Literature
	Background
	Alternatives
Static Context	Qualification required
	Training required
	Experience required
	Input (SE object)
	Output (SE object)
Applied in	SE phase

Context

Element	Attributes
Application domain	Industrial sector
Project	Size
	Kind of software
	Type of project
Environment	IT environment
	Development environment
	Development process
	Paradigm
	Interdependency

Impact

Element	Attributes
Impact on the dev. (project) costs	Training
	Introduction
	Application
	Maintenance
	Total cost of ownership
	Project
Impact on the quality of the product	ISO 9126
	Product
Impact on the development (project) schedule	Introduction
	Application
	Latency time
	Project
	Time to market
Latency in SE Phase	In SE phase

Figure 6: PCF Data Schema.

The overall resulting applicable development process can be summarized like the following suite of development phases, where the previous ones are required by the next ones: Aircraft Requirements Identification, Aircraft Function Development, Allocation of Aircraft Function to Systems, System Requirements Identification, Development of System Architecture, Allocation of System Requirements to Items, Item Requirements Identification, Item Design (corresponds to Software and Hardware Development, both having specific processes), Item Verification, System Verification, and Aircraft Verification.

These different phases can be well mapped onto the generic development model, by instancing the abstraction levels and by specifying the objectives of the viewpoints for each abstraction level, according to the company and project needs.

For example, at the system level, the System Requirements Identification corresponds to the Requirement Capture Viewpoint, the Development of System Architecture is realized via the Functional and Logical Viewpoints, the Allocation of System Requirements to Items belongs to the Technical Viewpoint, where the decision is taken on which technology will be involved to realized the Items (Item Design corresponds to Software and Hardware development). The Verification phases are realized in the Technical Viewpoint of corresponding abstraction levels, where the integration activity is performed. For each phase, objectives concerning safety assessments, validation, verification, etc. are defined via the Integral Process and should be met in order to move to the next phase, or must be accomplished during a next iteration. The same logic applies when moving to the next abstraction level.

Identical principles apply for all the other abstraction levels. This is also true for the Software and Hardware development, but with different steps inside the phases and different objectives, because they are defined by specific processes specified in the avionics standards DO-178C [2] and the DO-254 [19].

We consider the development of a safety-critical system – a Flight Control System (FCS). We give an example on how the regular avionic development process, according to the civilian aircraft and systems development process guidelines ARP4754A [12], can be mapped on the classification schema (see Figure 2). This mapping is shown in Figure 3, where the different represented process artifacts originate from the avionics V-Model depicted in Figure 1.

Here, we briefly introduce how to use the classification schema efficiently by describing the most important development process steps and their artifacts. The example summarized below starts at the system abstraction level. It follows the simplified process instance shown in Figure 3.

Based on the high-level aircraft requirements and design decisions, the requirements on the FCS must first be captured, expressed, and validated precisely (requirement viewpoint). The artifacts for this step are the functional and non-functional requirements that contain the goals of the system (e.g., “control the three axes of the aircraft: pitch, yaw, and roll”), the operational requirements (e.g., operational modes), the safety requirements (e.g., which criticality for which surface/axis), the high-level performance requirements (e.g., aircraft response time following cockpit control requests), etc. The requirement capture can be facilitated with model-based methods, for example by using context, use-cases and scenarios diagrams representable with SysML/UML diagrams and elements among other supported by the tools Enterprise Architect (Sparx Systems) or Artisan Studio (Atego), or with requirements tools using structured text, like with DOORS (IBM Rational) – see Figure 5, cell “System – Requirement”.

Once captured, the requirements must be validated, which is a transition criterion for proceeding to the next step. Different activities and requirements types are analyzed using different technologies, according to the avionics standards. For this step, manual reviews are performed.

These requirements, expressed as text, model or in-tools-integrated mix of both, are then considered as valid inputs for the design phase. Based on them, the behavior of the

system is analyzed and a functional architecture in the form of a network of the essential functions covering the major system functionalities must be formulated (functional viewpoint). An example of a major functionality at the system abstraction level is the altitude control via the pitch axis, which is realized by the elevator surfaces. Essential functions are those realizing the functionality and having an external interface with other parts of the system, for example actuator control, acquiring of the surface position, synchronization with the other surfaces, etc. For example, block definition diagrams from the SysML (e.g., with Enterprise Architect, Artisan Studio, …) and signal flow diagrams (e.g., with MATLAB/Simulink from The Mathworks) are suitable to model the functions network (Figure 5, cell "System – Functional"). The resulting functional architecture shapes a part of the outputs of this step. First simulations of the system overall behavior can be realized with MATLAB/Simulink and some SysML/UML tools supporting model execution (e.g., Artisan Studio). This contributes to an early system validation.

Once the definition of these functions and their related requirements is completed, a Functional Hazard Assessment (FHA) must be performed [12], still in the functional viewpoint, like shown in Figure 3. The resulting FHA requirements express a fundamental output required by the avionics process at the system design phase. The FHA produces safety requirements and design constraints for the next design step (inside the logical viewpoint) which are necessary to make decisions about the decomposition and structuration of the functions in order to realize a suitable system design. In the logical viewpoint (Figure 3, cell "System – Logical Viewpoint"), these essential functions are structured, completed, and/or decomposed in order to shape the components to be realized on this abstraction level – here named "logical components". The logical architecture determination is also efficiently supported by the SysML/UML technologies (block diagrams, activity diagrams) and tools, and the behavior can be well designed via control flow diagrams, state machines, etc., among other supported by MATLAB/Simulink (Figure 5, cell "System – Logical"). Both categories of artifacts serve the expression of the required output of the system design phase (system architecture, interfaces definition, behavior details). At the end of the logical viewpoint, different validation activities (part of the transition criteria) must be accomplished, like a Preliminary System Safety Assessment (PSSA), a preliminary common cause analysis (CCA), etc. [12] in order to validate the decisions made in the design phase, that is, in the functional and logical viewpoints. Simulation technologies (e.g., MATLAB/Simulink) can also be used to validate the interactions and behavior between the logical components, once they are correctly formalized.

Based on these components and their inherited requirements (the logical components are derived from the functions of the functional viewpoint, which are themselves derived from the requirements of the requirement viewpoint), technical solutions suitable for this abstraction level are identified or existing technical solutions are chosen (technical viewpoint, see cell "System – Technical Viewpoint" in Figure 3). These technical solutions are called "technical components" in this paper. The requirements expressed by the logical components drive the selection of the technical components. At the system (and equipment) abstraction level(s), the technical viewpoint contains the allocation activities like defined in the avionic process [12] and shown in Figure 1. Systematic methods and semi-automatic deployment tools can support the allocation activity. Common activity to all abstraction levels, the new developed, previously integrated or already existing technical components are integrated in the above abstraction level. These integrated components represent the major outputs of the technical viewpoint.

Iterations inside an abstraction level are feasible for introducing new requirements, or for increasing the reusability rate by considering already existing technical components. As a consequence, the structuring (decomposition and composition) of the logical components may be performed in a different way. A configuration management system is mandatory for managing the different alternatives and versions.

At the end of the technical viewpoint, different verification activities must be accomplished, depending on the abstraction level. At the system (and aircraft) one(s), a System Safety Assessment (SSA), a common cause analysis (CCA), etc. [12] are performed in order to verify the decisions made in the functional, logical, and technical viewpoints. These safety process verification activities are shown in Figure 3 and Figure 5 (e.g., cell "System – Technical Viewpoint"). For functional verification, generic tools and methods supporting these activities are very specific to the developed system (test bench, etc.). In some cases an incremental integration can be performed and parts of the system can be simulated with Model-in-the-Loop methods (e.g., with MATLAB/Simulink generated applications) to simplify the integration steps.

If the already existent technical components fulfill exactly the requirements expressed by the logical components mapped onto them, the work is completed and the associated requirements are considered as fulfilled. This is an ideal case of reusability and will probably not arise very often at higher abstraction levels such the Aircraft and the System levels, but may arise at the Equipment or Item level.

The technical components that do not exist yet or that do not completely fulfill the requirements expressed by the logical components mapped onto them, and the logical components that are still too complex to be allocated to a particular technical solution are both inputs for the next abstraction level. They express requirements that have not been fulfilled at the current abstraction level and must be dealt with at the next one. Thus, the work on the next abstraction level can start.

The traceability, required by avionics processes at the different abstraction levels, is performed 1) between the viewpoints of the same abstraction level and 2) between the abstraction levels. For this second case, the traceability is performed between the technical and logical viewpoints of a given abstraction level and the requirement viewpoint of the next abstraction level.

For example: For 1), the technical components (technical viewpoint) are assigned to the logical components (logical viewpoint) that drove their selection. For 2), on abstraction level AL, each technical component not already realized and each logical component that cannot be mapped to a technical component must be addressed on abstraction level AL-1. They express requirements to be captured in the requirement viewpoint of AL-1. The requirements expressed at the Requirement viewpoint of AL-1 are then linked to the requirements expressed by the corresponding technical and logical components from the abstraction level AL.

The other abstraction levels follow the same logic for each step with methodology objectives, process objectives and artifacts, and similar activities that need to be carried out. All of them can be well mapped in the classification schema.

For example, at the Aircraft abstraction level, similar process activities as for the system level are realized, like an FHA, a (Preliminary and final) Aircraft Safety Assessment ((P)ASA), and Common Cause Analyses (CCA). For the equipment abstraction level, Fault Tree Analyses (FTA) are required as well as Common Mode Analyses (CMA), etc. (see Figure 1 and Figure 3). At the item abstraction level, several different activities are also expected at the technical viewpoint, like the realization of hardware components or the implementation the software ones. Specifically to the software development, the avionics standard DO-178 [2] defines different phases (called "processes", such as the Software Requirements Process and the Software Design Process) with several objectives requiring numerous artifacts, such as requirements and detailed design descriptions, validation and verification artifacts, etc., which can be performed by using different methods and tools (e.g., for verification: Classification Tree, Equivalence Partitioning, Cause-and-Effect Analysis), each containing pros and cons, depending on the context of the current development. The selection of tools is specific to the company process implementation.

Another issue that belongs to the top-down process explained here is that the reusability of existing solutions potentially fulfilling parts of the system also requires suitable and standardized methods and tools. Existing technical solutions may also consist of components developed outside the company, such as microcontrollers, software libraries, etc. with other degrees of quality and using different processes. In any case, these existing solutions need to be completely and suitably characterized and must be integrated efficiently into the development process.

However, reusability is not a separate activity that can be transposed directly as a technology that can be integrated into the schema. In fact, it influences different activities, such as the decomposition in the design phase at the logical viewpoint, the accurate characterization of the existing solutions and the deployment activity at the technical viewpoint, etc. All these aspects related to reusability must also be taken into account in these activities. For example, it should be possible to integrate a systematic deployment process and its related techniques as explained by Hilbrich and Dieudonné [17] into the schema via these activities. As an example for this case, the software applications that are to be mapped optimally onto electronic execution units (ECU) need to be decomposed and structured in a way that makes them well compatible with the capabilities of the ECUs in order to allow the use of a minimum number of ECUs. However, on the other hand, the ECUs must be formalized completely and their description must be easily accessible by the system and software architects in order to influence the system design and to be correctly selected during deployment. In ARAMiS, we also provide a template for formalizing multicore processor capabilities in a form and on an abstraction level that can be used by system and equipment engineers. The formalization must be performed by the software and hardware engineers who design the ECUs. A noticeable advantage is to be able to validate per analysis or per simulation more aspects of the system, like the timing reactions, or the resource consumption.

These activities related to reusability are scattered across different cells of the matrix. At present, they need to be taken care of by the system designer. It would be helpful if they could be better integrated into the chain of methods and tools in the future.

V. THE EMPIRICAL STUDY

In this section, we first specify the goal of the study, describe the design used for the study and the procedure followed for its execution. Study design and material are described in deep so as to enable external replications of this study.

The main goal of this study is:

G1: to evaluate the RTP and PCF for the purpose of understanding their applicability to the technology selection process in the context of new product development in the avionic domain.

Since we are also interested to understand potential room for improvements, to adopt the framework in Liebherr-Aerospace, we also define a second goal as:

G2: to elicit the requirements for the next version of the RTP and PCF to be adopted in Liebherr-Aerospace.

A. Design and procedure

The focus group is designed as a group discussion to be executed in a timeframe of 2 hours with a set of participants (from 4 to 6) that provide their answers as group discussion.

The discussion is designed to gather information from the participants in regard to the following outcomes:

1. To gather the general feedback on the methodology
2. To understand the difficulties perceived in using the methodology
3. To understand if the methodology can help to save time
4. To elicit the requirements for the next version of the methodology

The study is planned as follows:

- 30 minutes introduction to the RTP and PCF (methodology)
- 40 minutes questions and answers
- 35 minutes: requirements elicitation

- 10 minutes: closing questions
- 5 minutes wrap-up

The discussion is driven by a session moderator, with experience in conducting empirical studies.

The questions raised by the moderator are:

Q1: Which is your general impression of the methodology?
Q2: Which difficulties do you see in using the methodology?
Q3: Which are the advantages and disadvantages in using the methodology?

After this first session of questions, the participants are asked to elicit the requirements for the next version of the platform by following these steps:

- Participants receive three post-its in three different colors (red, yellow and green), for a total of nine post-its.
  They are then requested to write the three most important features they would like to add (on the green post-its), remove (on the red post-its) or modify (on the yellow post its).
- Then, each participant is invited to describe what they wrote on the post-its.
- Finally, in group, participants are requested to group similar ideas.

Then, after the requirement elicitation, we conclude the session with the last 30 minutes of questions where we ask:

Q4: Do you think the methodology developed considering the requirements elicited, can be useful for your work?
Q5: Are you interested in using the methodology developed considering the requirements elicited, in the future?

VI. Empirical Study Results and Discussion

The study has been conducted on November 13 2014 from 8:30 to 10:35, respecting the planned time-frame of 2 hours.

Participants were 5 technical leaders for the development of flight control systems, from Liebherr-Aerospace. All participants were male, Germans, and have more than 5 years of experience in their position.

The technology has been introduced by one of the authors of the technology itself, working at Liebherr-Aerospace while the session has been moderated by a research assistant from the University of Kaiserslautern, expert in conducting and designing empirical studies.

A. General impression of the methodology

All participants had a positive impression but they requested more details to better understand it.

Q1: Which difficulties do you see in using the methodology?

One participant reported that they usually adopt a less structured process, starting from different point of the previously presented matrix. For this reason, he suggests to allow users to start in any point of the matrix, instead of in the first row and column. However, another participant made the remark, that the avionics development is a requirement based process, and it cannot be started in any development phase efficiently and such structure may be positive to avoid or limit the risks of rework. .

Two participants report that they use several standards that can influence the structure of the technology. A more detailed structure of the abstraction levels should be defined.

Q2: Which are the advantages and disadvantages in using the methodology at Liebherr-Aerospace?

Participants identified several advantages. The platform would provide a good overview of our process and the tools used. Moreover, the platform would allow to increase the quality of the development process, also helping to avoid to miss some steps.

Finally, the platform would increase the acceptance of some technologies, by means of the experience learnt from other groups.

Finally, they see some difficulties in applying this version of the platform to the current process applied at Liebherr-Aerospace, or this process has to be adapted.

B. Requirements elicitation for the next version of the platform

In order to understand if a new customized version of the platform should be developed, we now executed a task to elicit the requirements of the next version of the platform.

As introduced in the Study Design Section, participants received a total of 9 post its in 3 different colors and they were asked to individually write the 3 most important features they would like to add (on the green post-its), modify (on the yellow post its) or remove (on the red post-its).

We collected a total of 13 green post-its, 6 yellow and 1 red post-its.

After the first step, participants clustered the requirements in common groups.

The final groups identified are:

New Features (add):

- Definition of more precise viewpoints / more detailed for each step [4 participants]
- Definition of possible transitions between viewpoints [4 participants]
- Change Management support [1 participant]
- Problem Reporting [1 participant]
- Established preferred tools / solutions for each cell [1 participant]
- More standards inputs are needed [2 participants]

Changes :

- Separate the requirement column from the other columns [3 participants]
- Renaming Technical Viewpoint in “implementation” [1 participant]
- Change the strict separation of viewpoints into a more general one [2 participants]

Remove:

- Improve the graphical representation [1 participant]

C. Closing Questions

Before the beginning of this session, one participant had to left the focus group. We continued the session with the last two questions with 4 participants.

Q4: Do you think the methodology developed considering the requirements elicited, can be useful for your work?

All participants consider the methodology useful, considering the implementation of the requirements elicited.

Q5: Are you interested in using the methodology developed considering the requirements elicited, in the future?

All participants are willing to adopt the methodology in the future (considering the previously wished extensions).

D. Benefits

The classification schema provides benefits for different people working in software-related projects, especially for project managers, system and software engineers, and technology providers (software and hardware vendors).

The use case indicates that, from the point of view of engineers and decision makers, the classification schema provides an effective platform for searching for existing technologies. For industry domains strongly based on process based development, it also provides a toolbox for accurately specifying the use of each technology for rigorous process steps.

The main benefit for the ARAMiS project was that creating the classification schema for the avionics domain helped us to improve the schema. Several changes to the schema have been suggested based on issues raised during the application of the schema concept in practice. Another major benefit for the ARAMiS project was the identification and specification of methods and tools for improving the integration of multicore processors for safety-critical domains.

VII. Conclusions and Future Work

In this paper, we presented a use case reporting on the usage of a classification schema in the avionics domain and its implementation in the PCF tool and an empirical study, with the goal of evaluating the acceptance and elicit requirements for a future version of the schema and PCF.

The schema is aimed at collecting and integrating methods and technologies to support the activities of a structured development process. It allows decision makers to find the most appropriate technology based on the technologies interaction and integration on various levels to enable efficient design and development of complex systems.

The schema provides a matrix representation of the development activities classified into viewpoints and abstraction levels that enables users to easily search for the most appropriate technologies throughout the whole development lifecycle.

The empirical study has been conducted with five technical leaders for the development of flight control system, from Liebherr-Aerospace Lindenberg GmbH, that provided their answer so as to understand their acceptance and the applicability of the schema and its implementation in PCF in Liebherr-Aerospace.

Results of the empirical study show that the schema could be very useful in critical domains, such as avionic, and help process managers to enable knowledge transfer inside the company and keep track of the technologies used in previous projects and to maintain traceability throughout the whole process.

Future work includes the implementation of the recommendation collected during the focus group and the collection of existing technologies to create a baseline for the platform. Moreover, we are planning to run an empirical study to validate the effectiveness of the schema in different domains. .

Acknowledgment

This paper is based on research carried out in the ARAMiS and the SPES_XT projects, funded by the German Ministry of Education and Research (BMBF O1IS11035Ü, 01|S11035T, and BMBF 01|S12005K)

References

[1] D.Taibi, C. Plociennik, and L.Dieudonné, “A Classification Schema for Development Technologies,” Ninth International Conference on Software Engineering and Advances, IARIA, Oct. 2014, pp. 577-583, ISBN: 978-1-61208-367-4

[2] RTCA DO-178C, “Software considerations in airborne systems and equipment certification,” Dec. 2011.

[3] P. Reinkemeier, H. Hille, and S. Henkler, “Towards creating flexible tool chains for the design and analysis of multi-core systems,” Vierter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION 2020), colocated with Software Engineering 2014 conference, Feb. 2014. [Online]. Available from: http://ceur-ws.org/Vol-1129/paper37.pdf. Last access: 2014.07.21.

[4] ARAMiS project, “Automotive, railway and avionics multicore systems”. [Online]. Available from: http://www.projekt-aramis.de/. Last access 2014.07.18.

[5] R. Firth, W. G. Wood, R. D. Pethia, L. Roberts, and V. Mosley, "A classification scheme for software development methods,” Technical Report CMU/SEI-87-TR-041, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1987.

[6] V. Basili and D. Rombach, “Towards a comprehensive framework for reuse: A reuse-enabling software evolution environment,” Technical Report, University of Maryland, 1988.

[7] V. Basili and D. Rombach, “Support for comprehensive reuse,” Software Engineering Journal, vol. 6, Sep. 1991, pp. 303-316, ISSN: 0268-6961.

[8] V. Basili, G. Caldiera, and D. Rombach, “Experience factory,” In: Encyclopedia of Software Engineering, Marciniak, John J., Ed., New York: Wiley, pp. 469-476, 1994.

[9] A. Jedlitschka, D. Hamann, T. Göhlert, and A. Schröder, “Adapting PROFES for use in an agile process: An industry experience report,” Sixth International Conference on Product-Focused Software Process Improvement (PROFES 2005), Springer, Jun. 2005, pp. 502-516, ISSN: 0302-9743, ISBN: 3-540-26200-8.

[10] A. Ploskonos and M. Uflacker, “A classification schema for process and method adaptation in software design projects,” Tenth International Design Conference (DESIGN 2008), May 2008, pp. 219-228.

[11] K. Schwaber and M. Beedle, “Agile software development with Scrum,” Prentice Hall, 2002, ISBN: 0-13-067634-9.

[12] SAE ARP4754 Rev. A, "Guidelines for development of civil aircraft and systems," Dec. 2010. Available from: http://standards.sae.org/arp4754a. Last access 2014.07.21.

[13] K. Pohl, H. Hönninger, R. Achatz, and M. Broy, "Model-based engineering of embedded systems - The SPES 2020 Methodology," Springer, 2012, ISBN: 978-3-642-34614-9.

[14] SPES_XT project, "Software platform embedded systems". [Online]. Available from: http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html. Last access 2014.07.18.

[15] P. Diebold, L. Dieudonné, and D. Taibi, "Process configuration framework tool," Euromicro Conference on Software Engineering and Advanced Applications 2014, in press.

[16] P. Diebold, C .Lampasona, and D. Taibi, "Moonlighting Scrum: An agile method for distributed teams with part-time developers working during non-overlapping hours," Eighth International Conference on Software Engineering and Advances, IARIA, Oct. 2013, pp. 318-323, ISBN: 978-1-61208-304-9.

[17] V. Lenarduzzi, I. Lunesu, M. Matta, and D. Taibi, "Functional Size Measures and Effort Estimation in Agile Development: a Replicated Study," in XP2015, Helsinky, Finland 2015

[18] P. Diebold, "How to configure SE development processes context-specifically?," 14th International Conference on Product-Focused Software Process Improvement (PROFES 2013), Springer, Jun. 2013, pp. 355-358, ISSN: 0302-9743.

[19] RTCA DO-254, "Design Assurance Guidance for Airbone Electronic Hardware," Apr. 2000.

[20] R. Hilbrich and L. Dieudonné, "Deploying safety-critical applications on complex avionics hardware architectures," Journal of Software Engineering and Applications (JSEA), vol. 6, May 2013, pp. 229-235, ISSN: 1945-3124.

[21] K. Forsberg and H. Mooz, "The Relationship of System Engineering to the Project Cycle," First Annual Symposium of National Council on System Engineering, Oct. 1991, pp. 57-65.

Challenges in Assessing Agile Methods in a Multisite Environment

Harri Kaikkonen
Department of Industrial Engineering and Management
University of Oulu
Oulu, Finland
harri.kaikkonen@oulu.fi

Pasi Kuvaja
Department of Information Processing Sciences
University of Oulu
Oulu, Finland
pasi.kuvaja@oulu.fi

***Abstract*—Organizations utilize agile development methods in addition to multisite environments with the intent to reduce costs and development time. Assessment results and possible challenges of utilizing and adopting such methods are typically qualitative and lack concrete evidence of the challenges. An assessment survey instrument was used to analyze the transformation of a multisite software development organization from waterfall-type development into agile development. The transformation was done in two globally distributed sites in Finland and India. The assessment survey was conducted in the Finnish site 6 months after it had changed its working methods, and again 12 months later in both sites. The site in India had adopted similar methods after the previous assessment survey was conducted. The results of the assessment survey in the Finnish site indicated regression between the two assessment rounds, while the results in India appeared to be better compared to Finland in the second round. Analysis of the results suggests that cultural differences and time elapsed from the organizational transformation may influence the assessment results and should be taken into account when assessing the implementation of development methods.**

Keywords-organizational change; global software development; agile methods; Scrum; process assessment.

I. INTRODUCTION

This article extends previous work in [1]. Adopting agile development methods such as Scrum [2] and extreme programming (XP) [3] have seen a great deal of interest in the software development community because of their claimed benefits of delivering working software and being more responsive to changes, among other reasons [4]. However, scaling agile methods into larger organizations than those with a single or a few teams has come with some difficulties yet there have been several descriptions to address the matter (e.g., [5][6]).

As development organizations become larger, they often also spread out globally out of necessity or because of their business environments [7]. This, in turn, tends to cause a whole array of issues to be considered while managing the development work.

This article describes selected results of a quantitative process assessment conducted at a medium-sized software development organization. The organization adopted a Scrum-based software development process in their multisite organization. The adoption and the assessment were conducted in two phases. First, the process was adopted by a smaller unit in Finland with approximately 30 people. The unit was later assessed approximately six months after the adoption. With the experience gathered from the first site, similar processes were adopted in another site of about 50 people in India, within the same organization. The assessment was then repeated at both sites.

The aim of this article is to provide evidence of issues in assessing the implementation of organizational changes such as new development processes in a global software development (GSD) organization, or other multisite organization.

The remainder of the publication is organized as follows. Section II contains a description of related work as literature background of agile development methods and global software development. Section III presents a description of the assessment process and Section IV a description of the organization in which the assessment was conducted. Section V presents the relevant results of the assessments. Section VI includes discussion based on the results and the paper concludes in Section VII.

II. RELATED WORK

Several methodologies gained popularity in the late 1980's to early 1990's to challenge the prevalent waterfall development processes. Early methodologies include the spiral model [8] and incremental development methods [9]. Later, these methodologies and their influence spawned the agile movement to give more practical descriptions of how to develop software.

The agile movement gained publicity within the software development community during the 1990's, and was later epitomized in the agile manifesto, published in 2001 [4]. The manifesto was a collaborative agreement of what practitioners saw as the values and principles of agile software development. The original manifesto reads [4]:

"*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

In addition to the actual manifesto, the authors also described twelve principles behind it [4]. These twelve principles were agreed as common to the agile practitioners, although agile methods had already been described and were in use in many different settings. 'Agile methods' is an umbrella term for a wide different set of approaches (e.g., Scrum, XP and Kanban [10]), that have challenged the traditional waterfall model of software development and introduced a more lightweight process of producing software [11].

Today, Scrum can be viewed as the most widely used agile software development approach, but the first description of a 'Scrum team' was in an article about flexibility on traditional new product development [12]. In many ways, the software development waterfall model that Scrum challenges, can be characterized as similar to the stage-gate model [13] in physical product development - exactly what the Scrum-team was designed to challenge.

There are several defined roles, meetings, practices and working methods in the Scrum methodology. One example of typical practices in Scrum is the sprint retrospective. The retrospective is a meeting held after each sprint for inspecting and adapting the process and environment of development [14].

The main difference between Scrum and waterfall type of development is the iterative nature of development work. Key differences between all agile methods and traditional software development include iterative development and promoting empowered teamwork. However, a common misinterpretation of agile software development is that agility is achieved with practices and tools [11], although the focus should be on being agile, instead of doing it [4].

During the same time that agile methods started to become increasingly prevalent in software development, globalization of high-technology businesses have increased the need for GSD. Software and its use as both products and services has become a competitive weapon, which must be utilized efficiently to stay ahead in high-technology competition [7].

The factors that have accelerated this trend include [7]:

- the need to capitalize on the global resource pool to successfully and cost-competitively use scarce resources, wherever located;
- the business advantages of proximity to the market, including knowledge of customers and local conditions, as well as the good will engendered by local investment;
- the quick formation of virtual corporations and virtual teams to exploit market opportunities;
- severe pressure to improve time-to-market by using time zone differences in "round-the-clock" development; and
- the need for flexibility to capitalize on merger and acquisition opportunities wherever they present themselves.

According to [15], the benefits of GSD include reduced development costs, cross-site modularization of development work, access to large skilled labor pool, and closer proximity to markets and customers.

One of the most obvious benefits of GSD is reducing costs by moving development work into countries with lower wages. This is also the main driver for many companies to utilize GSD. However, distributing work among different locations reduces the cost savings by adding complexity to development projects with added communication interfaces [15].

Another benefit of GSD is the modularization of development work across sites, which is intended to reduce cycle-time of development. Modularization of work can cause integration issues since development work is separated and, therefore, needs to be addressed when designing modularization of work. Organizations have divided development work between sites based on features or development modules, but also created co-located or virtual teams that share the workload between sites [7][15].

It is also widely known that the skilled developers have a high impact on development speed and quality. Many organizations utilize GSD to access skilled labor pools in lower cost countries such as India or China when local resources are not abundant enough to fulfill the needs of organizations. Larger labor pools also provide for greater opportunities in scalability of the organization [7]. However, the attrition levels are higher in low-cost countries, which is a disadvantage in any development organization [15].

There are also benefits in locating developers closer to customers in some cases. By having developers with culturally and linguistically similar backgrounds as the customer, misunderstandings become rarer. Of course, the organization has to then find a balance between cultural divide internally and externally [15].

Although GSD is also intended to reduce time-to-market by utilizing "round the clock" development [7], as [15] has pointed out, the intended benefits of leveraging time-zone effectiveness and innovation and shared practice may not be that abundant in GSD. The differences in time-zones may not actually provide benefits for GSD and may, in fact, be a hindrance for development work. Time-zone separation reduces collaborative time window and may cause unusual working hours for both parties [15].

Also, the sharing of innovation and best practices among different locations may be problematic in some organizations. Employees who feel threatened by their lower cost colleagues may not always be willing to share all of their knowledge, which reduce the benefits of having best practices shared among the organization [15].

The challenges of GSD have been described in several sources (e.g., [15][16][17][18][19]). Issues range from strategic level issues such as how to divide work between different sites, to more tactical level problems like how to arrange effective daily communication channels, to more complex systems like cultural differences and their effect on project and process management [7]. It is clear that many types of issues become apparent when dividing any kind of work globally, and with development work that often realizes inside developers' and designers' heads, the problems can be all the more difficult. Methods have also been proposed to reduce the effects of the challenges involved with GSD. These methods range from the use of maturity models [20] to suggested practices and techniques [18].

As organizations try to improve their processes and products, they often turn to assessments to get further understanding of their processes. Some of these assessments have also been conducted in global development environments (e.g., [21]). Similarly to the identified challenges with GSD, analyzing assessment results from GSD organizations may also contain challenges that are unknown. This is true for assessment results in any multisite organization, not just for GSD organizations.

The challenges that have been described on GSD and on agile adoption are often based on interview studies and are qualitative by nature. There is a lack of quantitative assessment data on GSD environments and the effect of the challenges in the assessment data. The following sections provide an example of the issues that are related to assessing the implementation of agile methods in a multisite environment.

III. RESEARCH METHOD

One of the challenging things in any organizational transformation towards a new way of working is how to assess the transition process and guide the next steps. This research was conducted using the *Lean and Agile Deployment Survey*, which is an assessment instrument developed by the University of Oulu in collaboration with industrial partners in the Cloud Software Program [22] in Finland. The instrument is specifically designed for enabling an effective transformation to a lean and agile way of working. The survey is based on a generic structure of three organizational levels; portfolio, program and project [6], and focuses on four main dimensions: organizational set-up, practices, outputs and culture/mindset. The survey was part of a larger effort that University of Oulu was performing in identifying the right agile practices to adopt and to determine whether organizations are ready for lean and agile. Additionally, the approach is meant to provide information for deciding what necessary preparations and potential difficulties may be faced during the adoption process.

The conducted survey contained four context information questions for analyzing purposes, and over 70 statements that described the organization's agile development process as it had been planned and taken in use internally. The statements were tailored from general statements in the *Lean and Agile Deployment Survey* to correspond with the terminology and processes of the case organization. Some generic examples of the statements are presented below:

- *The product backlog prioritization is clear.*
- *The product owner guides the Scrum team by prioritizing the user stories.*
- *I understand when the user stories are complete and can be accepted within the sprint.*

IV. CASE ORGANIZATION

The case organization designs software for network protocol analyzers. One of the organization's sites in Finland started their agile transformation with pilots during the spring of 2010. They further changed that site's organization of around 30 employees to an agile way of working in the beginning of fall of the same year by starting to follow the methods of Scrum development [2][14]. During 2011, after initial results and experiences in Finland, similar processes were taken in use at a development unit in India and were planned to be taken in use in other sites as well.

The *Lean and Agile Deployment Survey* was conducted twice in the organization. The first survey took place about 6 months after the agile methods had been adopted in Finland. The second survey was conducted 12 months later and was expanded to include the site in India, which had adopted similar agile practices during that time.

The targets of the survey assessment were i) to review the current status of agile adoption at two of the case organization's sites, ii) to see how the unit in Finland had been progressing with agile methods between the two survey rounds, iii) to identify focus areas for continuous improvement efforts and iv) to receive feedback on the impressions and assumptions on agile and Scrum processes in other sites of the organization.

To obtain results for the last goal, the survey was also conducted in a third site, which had not yet fully adopted similar processes as the two case sites. The responses of the third site are omitted from the results presented in this publication.

The total number of respondents for the first round in Finland was 25. For the second round, there were 62 responses in total, 25 responses from Finland and 37 from India.

V. RESULTS

The survey was very successful in terms of response rate, which was a full 100 percent in the first round and 80.5 percent in the second round. The high response rate was

attributed to the close collaboration between the case organization and researchers and extensive communication to the survey participants. Personnel of the case organization also sponsored the survey noticeably, so it was well received.

TABLE I. RESPONDENT EXPERIENCE

	How many years of experience in software industry do you have?			
	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
None	0	0	0	0
Less than 2	1	0	6	6
2-5	4	2	13	15
5-10	5	5	16	21
10-20	13	16	2	18
More than 20	2	2	0	2
Total	25	25	37	62

A comparison of the respondents' experience shows that the personnel that participated in the survey were generally very experienced in software development (see Table I). There is also some difference between the experiences between the two sites. Many respondents in Finland had over a decade of experience in software development, which may amount to some opinions reflected in the survey results. The amount of experience in the Indian site is somewhat lower than in Finland, but, overall, the people in both sites had enough experience on software development to answer the survey.

TABLE II. RESPONDENT ROLES

	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
Developer	13	16	18	34
Tester	4	1	10	11
Product owner	4	4	2	6
Scrum master	3	2	6	8
Other	1	2	1	3
Total	25	25	37	62

Most of the responses in the survey came from developers and testers (see Table II). The other roles with significant number of responses were the product owner and Scrum master. As the focus of the survey was at the implementation of agile development process, the responses from these roles also provides a solid basis for the analysis of the results. The other roles that the respondents identified with were individual manager roles of the organization.

Because of the case organization's preference, the statements were evaluated by the respondents on a four-point scale, with an additional option of 'I don't know' instead of a 5-point Likert-type scale [23] usually utilized with the *Lean and Agile Deployment Survey*. The answering options with corresponding weights used in average calculation in the following results section were as follows (see Table III).

TABLE III. SURVEY ANSWERING OPTIONS

Option	**Option weight**
Disagree	1
Partially agree	2
Largely Agree	3
Fully Agree	4
I don't know	-

The following tables and figures present selected findings from the survey, which may be interesting in the context of multi-site agile adoption. The results for individual statements (see Figures 1-10) are presented as the distribution of responses and an average result in the statements in four separate rows. The first row presents the results received in the first survey that was conducted around 6 months after the agile adoption had taken place in the Finnish unit. The second and third rows include responses 12 months later from the Finnish and Indian units, respectively. The final row shows the combined answers in the second survey round from both sites (Finland and India).

Please note that the 'I don't know' –answers are not included in the average calculations. However, in some statements the amount of 'I don't know' –responses itself is significant.

Firstly, a very interesting finding can be made by looking at the collective average of the overall responses between the two survey rounds (see Table IV).

TABLE IV. SURVEY AVERAGE

	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
Response average	3,04	2,76	3,28	3,07

The fact that the average score in Finland in the second survey is lower than 12 months earlier is an alerting sign, as the statements were formed in a positive form in accordance with the case organization's process description (i.e., respondent's agreement with a statement would imply that the respondent feels that processes are followed as described). There was some indication from the case organization that they had not had sufficient resources to actively react to issues raised in the first survey and subsequent retrospectives during the 12 month period between the two surveys. One possible cause for the reduction in the average results may also be increased experience and awareness of the agile methods. This could

affect the results as people became more aware of their processes and the issues concerning them than before.

Perhaps surprisingly, the average score in India was much higher than it was in Finland as seen in the second round average scores. Several reasons may affect this difference, with cultural reasons perhaps being the most obvious explanation. The results of the first survey were also higher in Finland, although not as high as in Indian site's first respective survey. Naturally, lessons learned from implementing the practices in Finland first could have improved the implementation in India, which would yield more positive results in the latter implementation.

Reasons for the drop in score are evident in some survey results. One main improvement area for the case organization based on the first survey was the lack of identified value of continuous improvement activities (see Figures 1 and 2). Figure 1 shows how the respondents feel about the results of retrospectives. In the first survey round in Finland, it was identified that although the general view on retrospective results was positive, not all respondents felt that the teams were actually changing their ways of working based on retrospectives. The results of the same statement year after the first survey show actually even lower results, with none of the respondent's fully agreeing with this statement. There is also a small drop in the Finnish site's results on the statement "we reduce wasteful activities frequently" in the second survey round (see Figure 2). At the same time, the majority of Indian respondents show that the same practices are working well in their site.

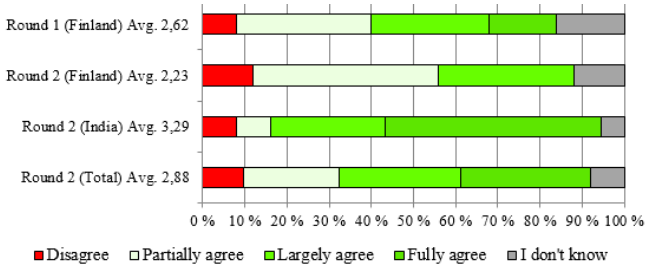


Figure 1. Scrum teams change their ways of working based on retrospectives.

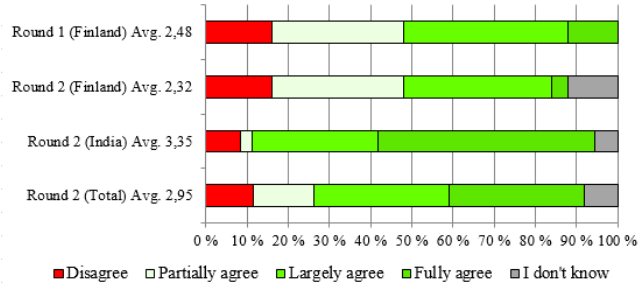


Figure 2. We reduce wasteful activities frequently.

The lack of resources assigned for following up on this improvement area show as reduced results in the topic of improving working practices. The results on continuous improvement are significantly higher in the Indian site with results averages being much higher.

A second major improvement area identified based on the first survey round was the lack of measured and communicated evidence of the benefits of the agile methods for the organization. This included the measuring and communication of benefits in productivity (see Figures 3 and 4), product quality (see Figure 5) and development time (see Figure 6).

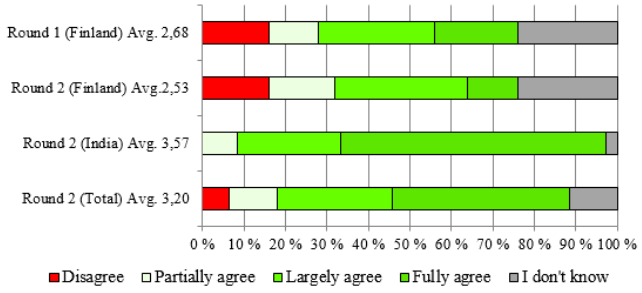


Figure 3. I am more productive with the agile way of working.

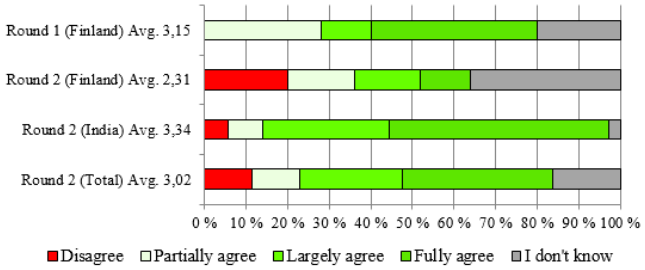


Figure 4. We are more productive as a Scrum team.

In the first survey round in Finland, it was evident that there were at least some differences of opinion in individual productivity between previous waterfall type of development and newly introduced agile methods (see top of Figure 3). There were also respondents who were unaware of any productivity changes related to their individual performance. Surprisingly, there were no disagreeing opinions on the productivity increase for the entire team even though some felt that their individual performance had decreased at the same time. This would imply that even if some individuals were not convinced of increases to their individual productivity, they felt that as a team their performance had still improved. Even if this was the case in the first survey round in Finland, in the second survey round the results had also dropped dramatically in terms of team productivity.

Again the results in this topic appear very high in India in relation to Finland. Especially in individual productivity (see Figure 3), the responses in India are overwhelmingly positive. Similar trend is evident also in the results of team productivity.

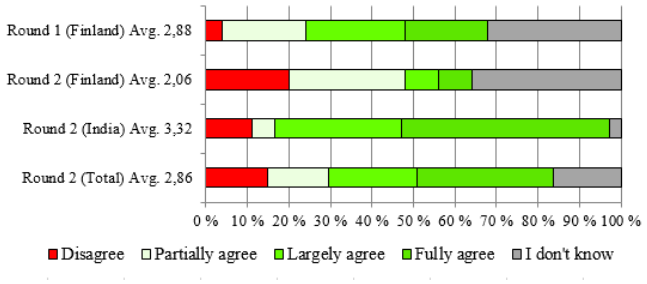


Figure 5. Product quality has been improved by applying agile development.

In Finland, similar trends are seen in the results of perceived improvements in product quality as in performance (see Figure 5). There are a lot of 'I don't know' responses in Finland in the statement "Product quality has been improved by applying agile development". This was the case in both survey rounds, but the results average in this statement also decreased between the survey rounds similarly to the previous statements about productivity. In India, the results are not as positive in terms of quality as they are in productivity, although the results are again much higher than they are in Finland.

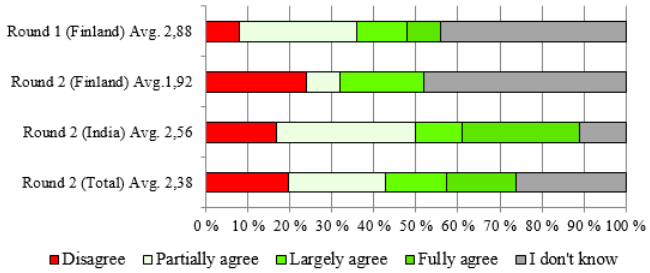


Figure 6. Development time has decreased by applying agile development.

Similar trends as with productivity and product quality changes are evident in yet another topic on development time (see Figure 6). In the first survey round, there was a lot of 'I don't know' responses to the statement 'Development time has decreased by applying agile development'. This was also the case in the second survey round, but many respondents also plainly disagreed with this statement. Actually, in the second survey round, only a handful of people showed any level of agreement to this statement and most of the respondents either disagreed with the statement or did not answer to it.

In India, there were also some 'I don't know' responses and disagreement with the above statement. However, a majority agreed with the statement saying that the development time has actually decreased with the introduction of agile methods.

An action point after the first survey round was to provide the teams more information on the benefits of agile in comparison to earlier working methods. This issue had apparently not received enough attention because the second survey round indicated some decrease in all results on the matter as well as an increase in 'I don't know' –responses in Finland. Another possibility for the results is that the quality and productivity have actually not been improved with the new methods. The measuring of the benefits of agile is a very interesting and difficult topic among all organizations implementing the methods, but high consideration should be used on how to provide teams more information on actual benefits of agile.

Other findings of the survey showed that there were also possible needs for further training within the organization (see Figures 7 and 8).

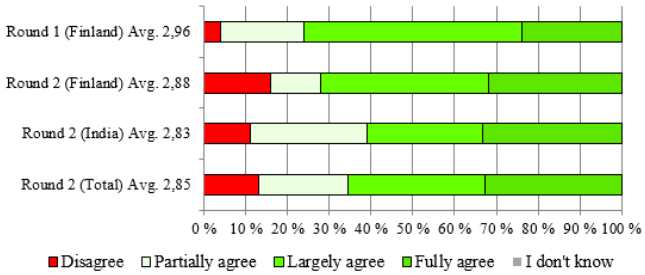


Figure 7. I have received enough training for carrying out my work.

In the responses of individual competence, there were small disagreement with the statement 'I have received enough training for carrying out my work. Even though most of respondents appear to be satisfied with their training, it may be interesting to note that the results in Finland dropped between the two survey rounds to include more respondents disagreeing with this statement. As the nature of the work during the 12 months between the surveys did not change, this may be indication of "conscious incompetence", where people become more aware of their own needs for more training with time. In contrast to the other findings, in this particular statement there were no major differences in the responses between Finland and India.

Even though there were no significant differences in the statement about training needs, there were some differences between the sites in terms of individuals feeling confident about themselves with the agile way of working (see Figure 8). In the first survey round in Finland, nearly all respondents largely or fully agreed with the statement 'I feel confident with myself with the agile way of working'. In the second survey round, the results in this statement had dropped significantly even though the methods had been in use longer and, therefore, should be more familiar to the respondents. In India, the distribution of responses appear to be very similar than in Finland in its respective first round. The respondents in India feel very confident with themselves with the new working methods.

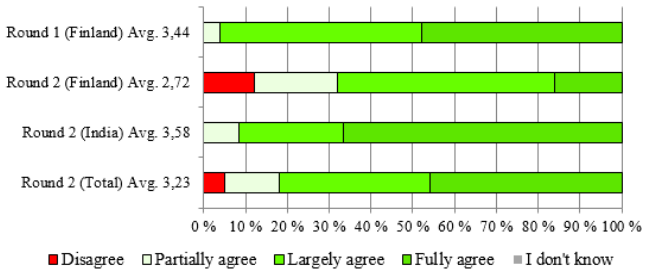


Figure 8. I feel confident with myself with the agile way of working.

When comparing the results between the sites in Finland and India, it can be seen that the training needs appear to be equally divided between the two sites. However, there is a noticeable difference between the sites in the confidence in individual capabilities. One possible explanation for this can be cultural differences in answering the survey, but as the results appear similar in both sites with their respective first implementation of the survey, this could imply that the results will reduce with time.

Perhaps the most compelling evidence of bias in the survey results is evident in statements about the preference of team co-location between the sites (see Figures 9 and 10). There is a very noticeable difference in the answers between Finland and India.

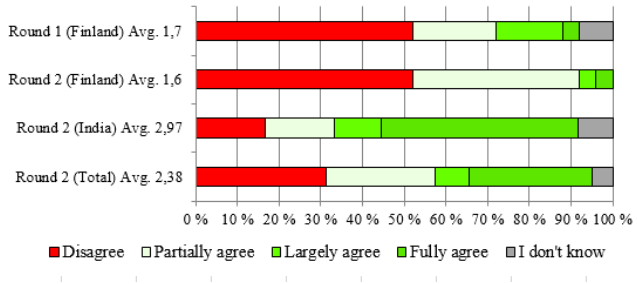


Figure 9. I prefer to work in a multisite Scrum team.

When asked about the preference of working in a multisite Scrum team (see Figure 9), the results in Finland remain similar between the two survey rounds. Most of the respondents disagree with the statement 'I prefer to work in a multisite Scrum team', and there are only individual respondents fully or largely agreeing with the statement. In the second survey round, the results have reduced even more with most of the agreeing responses being now only in the 'partially agree' –response. In India, the results in the same statement appear somewhat different with almost half of the respondents fully agreeing with this statement. There are also some disagreeing opinions to the statement and individual 'I don't know' –responses.

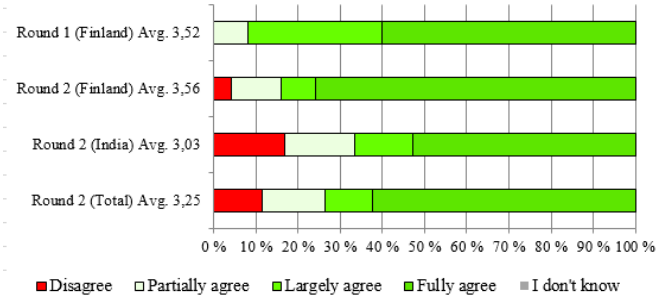


Figure 10. I prefer to work in a local Scrum team.

When asked about the preference of location the other way around, the responses show distinctive differences between the two sites (see Figure 10). In the statement 'I prefer to work in a local Scrum team', the responses in Finland appear to be very parallel to the previous statement on preferring to work in a multisite Scum team as expected. This was the case in both survey rounds, and the amount of 'fully agreeing' responses to this statement formed the clear majority.

On the contrary, in India the responses to this statement of local Scrum team preference were not parallel to the previous statement, or actually anywhere near it. The responses were mostly agreeing in both statements of team location preference, which in nature should be contrary with each other. The distribution of responses between these two statements is also very similar, with most respondents 'fully agreeing' in both statements.

Differing from the answers in India, there seems to be a clear preference to co-location of team members in Finland. The co-location in generally viewed as an important part in Scrum processes and the results in Finland show the preference that has come by experience in that site. The conflictingly high results of India in both of the two above tables may involve cultural influences, but also some lack of experience since the agile methods had been in use there for a shorter period of time.

It should be noted that the statements on team location preference (Figures 9 and 10) were asked next to each other. During the analysis, the results of these two statements also raised the question of whether some respondents did not bother reading the survey statements at all. The validity of these responses was ensured by looking at individual response sheets and there was no evidence of individuals not filling the survey purposefully.

Although generalizing a large set of survey statements may be problematic, an additional interesting comparison was made between the two survey rounds in the overall amount of 'I don't know' –answers (see Table V).

TABLE V. PERCENTAGE OF 'I DON'T KNOW' RESPONSES IN ALL STATEMENTS

Round 1 (Finland)	*Round 2 (Finland)*	*Round 2 (India)*	*Round 2 (Total)*
9,8%	12,7%	7,62 %	9,8%

In the second survey round, the amount of 'I don't know' -answers in Finland is quite a lot higher than in India. When results between the two rounds are compared, we find that the percentage in Finland has increased between the two rounds and that the percentage in India is even lower than in Finland in the first round. There was a similar amount of time elapsed from the agile adoption in Finland in the first round and India in the second. This could indicate that the amount of knowledge acquired during the 12 months between the survey rounds in Finland had lead to an increase in awareness over issues.

This can also be an indication of the cultural differences in the two locations. The total percentage of 'I don't know' –responses appeared to be identical between the two survey rounds at first, but the difference in the amount between the two sites is a clear indication that this type of analysis should be done to get a comprehensive image of survey results.

VI. DISCUSSION

Generally speaking, the case organization was pleased with the results obtained by the adoption of agile methods in their development process. They felt that agile methods had increased communication and predictability of their software processes, even if there had been some difficulties in the adoption. The results highlighted in this publication did not diminish the organization's commitment to the agile methods and the survey results were openly communicated throughout the assessed organizational units.

Based on the survey results, the main improvement areas identified in the first survey round were not given enough attention after conducting the survey. This was also admitted by the case organization because of reduced resources for the improvement efforts. This is one of the main reasons why the results in the Finnish site appear lower in the second round.

As an organization commits itself to new working methods, it also has to show and communicate this commitment. In this case organization, there was a clear lack of resources for improving the newly introduced methods, which shows as apparent lack of commitment from the employees as well as time passed. If an organization does not provide sufficient resources for change management and addressing possible issues that arise from organizational changes, the employees may lose confidence in the changes and start reverting to old working methods.

Another main reason for the reduction in response averages in some statements is believed to be increased awareness on the topic of agile methods and possible issues related to them. The combined average result in all statements between Finland in round 1 and India in round 2 are similar. The amount of time that these two sites had been using the agile development methods before their first respective survey rounds was also similar. This would indicate that as the adoption methods of the case organization were similar in the two sites, the results of the working practices appear to be the same in the beginning of the adoption for the most part. However, the increase in the amount of 'I don't know' –responses in Finland over time may be indicative of increased awareness of the process and that it is not followed as it should.

Based on the first survey results, the first important improvement suggestion for the case organization was to improve the resources currently utilized for change management and improvement efforts. It was suggested that the teams may need more support and resources for successful organizational transformation. The reduction in the results related to improvement efforts (see Figures 1 and 2) shows that these suggestions were not followed sufficiently.

The same suggestion was given to the case organization after the second survey round as well. It was noted that the reduction in the results regarding improvement efforts must be given higher priority in both sites. In Finland, more resources were to be made available to start addressing possible issues with the new working methods to ensure that the employees will stay committed to the changes. In India, same resources should be made available to prevent a similar reduction in the results, if possible. It may be that the previously mentioned notion of increased awareness and subsequent reduction of results will become evident anyway. However, this should not diminish the importance of the organization communicating its commitment to the organizational changes.

The increased resources should include both support for continuous improvement activities and especially the follow up of these activities, since there were no definitive improvements that could be identified from the first assessment round.

The identified decrease in results should be taken seriously to see what kind of improvement actions could be taken. This should also include very active participation from all members of the development organization, since they will be most aware of the issues regarding their daily work. The practices and processes that do not work should be adapted according to the organization- or unit-specific preferences while remembering to include the agile principles and mindset.

Continuous improvement activities should have a strict process to follow, which includes communication to all interested stakeholders on the progress of the activities and responsible individuals who have allocated time to conduct the activity. [24] has also presented evidence of additional success factors that can support the sustainability of improvement activities as well, which should be kept in mind when implementing changes.

The follow-up of the activities should also include a larger scale follow-up of the adoption of agile methods. Some forms of quantitative or qualitative measurements of the possible benefits of agile (in productivity, quality, etc.) should be measured and communicated in all units, including the sites that may take the agile methods into use in the future. This shows that the organization is committed to the changes and that the activities that are requested of the members of the organization have justifications behind them. There was already some evidence of doubt in the agile methods in the first survey round and these doubts should be addressed properly through discussion.

Measuring the benefits of organizational or process changes can be difficult, especially if these kinds of metrics are not introduced before the changes. The adoption of organizational changes will still have some motivation, and that motivation needs to be assessed after the changes have been done in order to make sure that the changes have been valid.

In addition to the assessment results changing with time elapsed between the organizational change and the assessment, the results of the survey also indicate bias in the results based on cultural differences. When assessing the success of multisite organizational changes, it should be noted that the results may vary between locations for reasons that may not be possible to influence with any change management processes. Therefore, it may be useful in some cases to assess different global sites individually, instead of comparing the results of sites between each other.

The cultural differences in this article have been identified on a national level. The differences can be also found on other levels of the organization as well. Differences between working cultures also appear on individual and team levels, and no individuals should be categorized only based on their nationality or other characteristic. However, based on this survey, there are cultural characteristics, which need to be taken into account when assessing organizational changes. Any assessment effort related to organizational changes should be made with the notion that there may be underlying differences between nationalities, sites, organizational levels or working cultures.

VII. CONCLUSION

The managerial implication of this study is the importance of providing resources for following up on continuous improvement activities when they are expected from the employees. If resources are not available for removing issues or attention is not given to the improvement tasks, employees may also start losing confidence in organizational changes. The other managerial implication is the high importance of measuring and communication of the benefits of organizational changes. Without clear communication about the possible benefits of organizational changes, employees will start to doubt their significance, which will hinder further improvement efforts as well.

Scientific implications of this study is the importance of taking cultural factors into consideration when analyzing research results. The evidence on the influence of cultural factors does is not limited to assessments such as [21], but also maturity models such as [20]. Cultural influences may cause issues in comparison between research results.

The results of this research can be used by researchers and practitioners when assessing organizational changes. Assessment results between geographically distributed sites may not always be directly comparable between each other. Cultural differences in results and the difference in elapsed time from the organizational change may also affect assessment results and should be noted when analyzing data.

It would also be beneficial to compare results of a similar assessment with a different scaling method like, e.g., the Likert-type scale. The scaling itself should not be a contributing factor in this study, but additional assessment cases with similar backgrounds could be used to validate the influence of the used survey scale.

The assessment process could be repeated in the case organization for a third time to analyze further progress of the organizational change. The findings of this assessment were used to focus future improvement efforts in the case organization and to provide feedback on how they understand their agile transformation so far. The results were presented to all participants through an open discussion session by the researchers and a written report was communicated openly inside the organization. The report was also brought into general knowledge by giving access to it within the organization.

The findings of this study related to cultural influences affecting assessment results should also be validated in other assessments. The results should be applicable in other types of assessments as well, but this would need further validation of the results.

ACKNOWLEDGMENT

The authors would like to acknowledge the contribution of the Finnish Cloud Software Program [22] for the funding received for this research.

REFERENCES

[1] H. Kaikkonen, P. Rodriguez, and P. Kuvaja “On Some Challenges in Assessing the Implementation of Agile Methods in a Multisite Environment,” The Ninth International Conference on Software Engineering Advances (ICSEA 2014) IARIA pp. 179-184, ISBN:978-1-61208-367-4

[2] K. Schwaber and M. Beedle, “Agile Software Development with SCRUM,” Prentice Hall, 2001.

[3] K. Beck, “Embracing change with extreme programming,” Computer, Volume 32, Issue 10, pp. 70-77, 1999.

[4] K. Beck et al., “Principles behind the agile manifesto,” [Online] Available from: http://agilemanifesto.org/principles.html 2014.08.12.

[5] C. Larman and B. Vodde, “Scaling Lean & Agile development. Thinking and Organizational tools for Large-scale Scrum,“ Addison-Wesley, USA. 2009.

[6] D. Leffingwell, "Scaling software agility: Best practices for large enterprises," Pearson Education, USA, 2007.

[7] J.D. Herbsleb and D. Moitra, "Global software development," IEEE Software, Volume 12, Issue 2, pp. 16-20, 2001.

[8] B.W. Boehm, "A spiral model of software development and enhancement," Computer, Volume 21, Issue 5, pp. 61-72, 1988.

[9] C. Larman and V. Basili, "Iterative and incremental development: a brief history," Computer, Volume 36, Issue 6, pp. 47-56, 2003)

[10] D. Anderson, "Kanban - Successful Evolutionary Change for your Technology Business," Blue Hole Press, USA, 2010.

[11] M. Poppendieck and M. Cusumano, "Lean software development: A Tutorial," IEEE Software, Volume 29, Issue 5, pp. 26-32, 2012.

[12] H. Takeuchi and I. Nonaka, "The new new product development game," Harvard Business Review, January-February 1986, pp. 137-146, 1986.

[13] R.G. Cooper "Winning at new products," Addison-Wesley, Mass., USA, 1986.

[14] P. Deemer, G. Benefiend, C. Larman, and B. Vodde, "Scrum primer v2.0. A lightweight guide to the theory and practice of Scrum," [Online] Available from http://www.infoq.com/resource/news/2013/02/scrum-primer-book-download/en/resources/ , 2012. 2015.02.09

[15] E.Ó. Conchúir, P. J. Ågerfalk, H.H. Olsson, and B. Fitzgerald, "Global Software development: Where are the benefits?," Communications of the ACM, Volume 52, No. 8, 2009.

[16] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," The Journal of Systems and Software, Volume 83, Issue 1, pp. 67-76, 2010.

[17] P.J. Ågerfalk et al., "A framework for considering opportunities and threats in distributed software development," Proceedings of the International Workshop on Distributed Software Development, Paris, France. Computer Society, 2005. pp. 47-61

[18] A. S. Alqahtani, J. D. Moore, D. Harrison, and B. Wood, "Distributed agile software development challenges and mitigation techniques: A case study," The Eight International Conference on Sofware Engineering Advances, (ICSEA 2013) IARIA pp. 352-358, ISBN: 978-1-61208-304-9.

[19] A. Mockus and J. Herbsleb "Challenges of Global Software Development," Proceedings of the Seventh International Software Metrics Symposium, (METRICS 2001, IEEE), pp. 182-184.

[20] T. Oliveira and M. Silva, "Method for CMMI-DEV Implementation in Distributed Teams," The Sixth International Conference on Software Engineering Advances (ICSEA 2011) IARIA pp. 312-317, ISBN:978-1-61208-165-6.

[21] S. Misra and L. Fernández-Sanz, "Quality Issues in Global Software Development," (ICSEA 2011) IARIA pp. 325-330, ISBN: 978-1-61208-165-6.

[22] Cloud Software Finland. [Online]. Available from: http://www.cloudsoftwareprogram.org/cloud-software-program 2014.08.12.

[23] R. Likert. "A Technique for the Measurement of Attitudes," Archives of Psychology Volume 140, pp. 1–55, 1932.

[24] N. Nikitina and M. Kajko-Mattson, "Success factors leading to the sustainability of software process improvement efforts," The Sixth International Conference on Software Engineering Advances (ICSEA 2011) IARIA pp. 581-588, ISBN: 978-1-61208-165-6.

Automated Unit Testing of JavaScript Code through Symbolic Executor SymJS

Hideo Tanida, Tadahiro Uehara

Information System Technologies Laboratory
Fujitsu Laboratories Ltd.
Kawasaki, Japan
Email: {tanida.hideo,uehara.tadahiro}
@jp.fujitsu.com

Guodong Li, Indradeep Ghosh

Software Systems Innovation Group
Fujitsu Laboratories of America, Inc.
Sunnyvale, CA, USA
Email: {gli,indradeep.ghosh}
@us.fujitsu.com

***Abstract*—JavaScript is expected to be a programming language of even wider use, considering demands for more interactive web/mobile applications and deployment in server-side software. While reliability of JavaScript code will be of more importance, testing techniques for the language remain insufficient, compared to other languages. We propose a technique to automatically generate high-coverage unit tests for JavaScript code. The technique makes use of symbolic execution engine for JavaScript, and symbolic stub/driver generation engine, which injects symbolic variables to system under test. Our methodology allows for automatic generation of input data for unit testing of JavaScript code with high coverage, which ensures quality of target code with reduced effort.**

Keywords–JavaScript; test generation; symbolic execution; symbolic stub and driver generation.

I. INTRODUCTION

Extensive testing is required to implement reliable software. However, current industrial practice rely on manually-written tests, which result in large amount of effort required to ensure quality of final products or defects from inadequate testing.

Verification and test generation techniques based on formal methods are considered to be solutions to overcome the problem. One such technique is test generation through symbolic execution, which achieves higher code coverage compared to random testing [1]–[7].

In order to symbolically execute a program, input variables to the program are handled as symbolic variables with their concrete values unknown. During execution of the program, constraints to be met by values of variables in each execution path are obtained. After obtaining constraints for all the paths within the program, concrete values of input variables to execute every paths can be obtained, by feeding a solver such as Satisfiability Modulo Theory (SMT) [8] solver with the constraints. Normal concrete execution of the program using all the obtained data, results in all the path within the program went through.

Manually-crafted test inputs require effort for creation, while they do not guarantee exercising all the execution path in the target program. In contrast, test generation based on symbolic execution automatically obtains inputs to execute all the path within the program. As the consequence, it may find corner-case bugs missed with insufficient testing.

There are tools for symbolic execution of program code, including those targeting code in C/C++ [2], [3], [5], Java [4], and binary code [6], [7]. It is reported that the tools can automatically detect corner-case bugs, or generate test inputs that achieve high code coverage.

JavaScript was historically introduced as an in-browser scripting language of light weight use. However, it is now heavily used for implementation of feature rich client-tier within interactive web applications. The language is also used to implement software product of other kind, including application servers based on Node.js [9] and standalone mobile applications implemented with PhoneGap [10].

The wider adoption of the language has brought efficient testing technique for JavaScript code into focus. In order to exercise tests in an efficient manner, unit testing frameworks such as Jasmine [11], QUnit [12] and Mocha [13] are developed and used in the field. However, only execution of once developed tests can be supported through the tools, and large amount of effort is still required to prepare test cases that ensure quality of target code. Therefore, automatic test generation techniques for the language are becoming of more importance, and symbolic execution is again considered one key technology to play a role.

Existing symbolic execution tools for JavaScript code include Kudzu [14] and Jalangi [15]. Kudzu automatically generates input data for program functions, with the aim of automatically discovering security problems in the target. Jalangi allows for modification of path constraints under normal concrete executions, in order to obtain results different from previous runs. However, the tools could not be applied to unit testing of JavaScript code in field, due to limitations in string constraint handling and need for manual coding of driver and stub used in testing.

We propose a technique to generate test inputs for JavaScript code through symbolic execution on a tool SymJS [1], [16]. Test inputs generated by the tool exercise target code with high coverage. After augmenting generated test inputs with user-supplied invariants, application behavior conformance under diverse context can be checked in a fully automatic fashion. Our proposal includes automatic generation of symbolic stubs and drivers, which reduces need for manual coding. Therefore, our technique allows for fully automatic generation of input data used in unit testing of JavaScript code. Test inputs generated by our technique exercise every feasible

execution paths in the target to achieve high coverage.

Our methodology has the following advantages to existing works. Our JavaScript symbolic execution engine SymJS is applicable to JavaScript development in practice for the following reasons.

- Our constraint solver PASS [17] allows test generation for programs with various complex string manipulations
- SymJS does not require any modification to the target code, while the existing symbolic executors for JavaScript [14], [15] needed modifications and multiple runs
- Our automatic stub/driver code generation allows for fully automatic test data generation

An existing work [15] could be employed for generation of unit tests. However, it required manual coding of stub/driver, which requires knowledge on symbolic execution and error-prone. The engine also suffered from limitations in string constraint handling. Our fully automatic technique can be applied to development in practice.

The rest of this paper is organized as follows. Section II explains the need for automatic test generation/execution with an example, and introduces our test input generation technique through symbolic execution. Section III describes our method to automatically generate stub/driver code used in test generation/execution. Evaluation in Section IV shows applicability and effectiveness of our technique on multiple benchmarks. We discuss the lessons learnt in Section V. Finally, we come to the conclusion in Section VI and show possible future research directions.

II. Background and Proposed Test Generation Technique

A. Demands for Automatic Test Generation

Generally, if a certain execution path in a program is exercised or not, depends on input fed to the program. Therefore, we need to carefully provide sufficient number of appropriate test input data, in order to achieve high code coverage during testing.

For example, function `func0()` shown in Figure 1 contains multiple execution path. Further, whether each path is exercised or not depends on input fed to the program, which are value of arguments `s,a` and return value of function `Lib.m2()`. Current industrial testing practice depends on human labor to provide the inputs. However, preparing test inputs to cover every path within software under test requires large amount of efforts. Further, manually-created test inputs might not be sufficient to exercise every path within the target program.

Figure 2 shows possible execution path within the example in Figure 1. In the example, there are two sets of code blocks and whether blocks are executed or not depend on branch decisions. The first set of the blocks contains blocks 0-3, and the second set contains blocks A-B. Conditions for the blocks to be executed are shown at the top of each block in Figure 2. Block 2 has a contradiction between conditions for

```
function func0(s,a) {
  if("".equals(s)) { // block 0
    s = null;
  } else {
    if(s.length <= 5) { // block 1
      a = a + status;
    } else {
      if("".equals(s)) { // block 2
        Lib.m0(); // Unreachable
      } else { // block 3
        Lib.m1();
      }
    }
  }
  if(a <= Lib.m2()) { // block A
    a = 0;
  } else { // block B
    a = a + s.length; // Error with null s
  }
}
```

Figure 1. Code Fragment Used to Explain our Methodology: s, a, Lib.m2() may Take Any Value

Table I. Constraints to Execute Paths in Figure 2 and Satisfying Test Inputs (under assumption status=-1)

Test No.	Blocks Executed	Path Conditions	Test Data
1	0,A	`"".equals(s)` ∧ `a<=Lib.m2()`	`s="", a=0` `Lib.m2()=0`
2	0,B	`"".equals(s)` ∧ `a>Lib.m2()`	`s="", a=0` `Lib.m2()=-1`
3	1,A	`!"".equals(s)` ∧ `s.length <= 5` ∧ `a-1<=Lib.m2()`	`s="a", a=0` `Lib.m2()=0`
4	1,B	`!"".equals(s)` ∧ `s.length <= 5` ∧ `a-1>Lib.m2()`	`s="a", a=1` `Lib.m2()=0`
5	3,A	`!"".equals(s)` ∧ `s.length > 5` ∧ `a<=Lib.m2()`	`s="aaaaaa", a=0` `Lib.m2()=0`
6	3,B	`!"".equals(s)` ∧ `s.length > 5` ∧ `a>Lib.m2()`	`s="aaaaaa", a=0` `Lib.m2()=-1`

execution, which are `s.length()>5` and `"".equals(s)`, and will never be executed. However, the other blocks have no such contradiction and are executable. Tests to execute every possible combination of blocks 0,1,3 and blocks A,B correspond to $3 \times 2 = 6$ set of values for the inputs.

Table I shows combinations of blocks to execute, path condition to be met by arguments `s,a` and return value of `Lib.m2()`. In the example, it is possible to obtain concrete values meeting the conditions for the inputs, and the values can be used as test inputs. We will discuss how to automatically obtain such test inputs in the sequel.

B. Test Input Generation through Symbolic Execution

We propose a methodology to automatically generate test inputs with SymJS, a symbolic execution engine for JavaScript. During symbolic execution of a program, constraints to be met in order to execute each path (shown as "Path Conditions" in

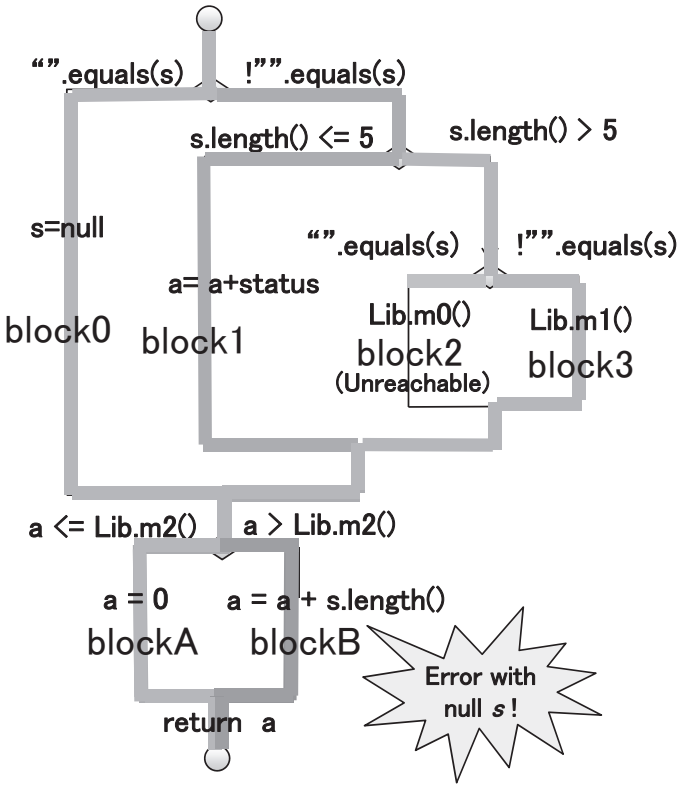


Figure 2. Execution Paths within Code Shown in Figure 1

Table I) are calculated iteratively. After visiting every possible path within the program, constraints corresponding to all the path are obtained. Concrete values of variables meeting the constraints can be obtained with solvers such as SMT solver. Obtained values are input data to exercise paths corresponding to the constraints, which we can use for testing.

While JavaScript functions are often executed in a event-driven and asynchronous fashion, our technique focuses on generation of tests that invoke functions in deterministic and synchronous orders. We assume the behavior of generated tests are reasonable, considering what is inspected in current JavaScript unit tests in field, as opposed to integration/system testing. Each generated test data exercise single path within the target, and only single data set is generated for each path.

SymJS allows for symbolic execution of JavaScript code. SymJS interprets bytecode for the target program, and symbolically executes it in a way KLEE [3] and Symbolic JPF [4] do. SymJS handles program code meeting the language standard defined in ECMAScript [18].

SymJS is an extended version of Rhino [19], an open-source implementation of JavaScript. Our extensions include symbolic execution of target code, constraint solving to obtain concrete test input data, and state management. While there are existing symbolic executors for JavaScript, SymJS does not reuse any of their code base. Table III shows comparison between SymJS and existing symbolic executors.

SymJS interprets bytecode compiled from source code of target program. This approach is taken by existing symbolic executors such as KLEE [3] and Symbolic PathFinder [4].

Table II. Instructions with their Interpretations Modified from Original Rhino

Arithmetic/Logical Operations	ADD, SUB, MUL, DIV, MOD, NEG, POS, BITNOT, BITAND, BITOR, BITXOR, LSH, RSH, URSH etc.
Comparisons	EQ, NE, GE, GT, LE, LT, NOT, SHEQ, SHNE etc.
Branches	IFEQ, IFNE, IFEQ_POP etc.
Function Calls	RETURN, CALL, TAIL_CALL etc.
Object Manipulations	NEW, REF, IN, INSTANCEOF, TYPEOF, GETNAME, SETNAME, NAME etc.
Object Accesses	GETPROP, SETPROP, DELPROP, GETELEM, SETELEM, GETREF, SETREF etc.

Handling bytecode instead of source code allows for implementation of symbolic executors without dealing with complex syntax of the target language. SymJS is implemented as an interpreter of Rhino bytecode, which updates the program state (content of heap/stack and path condition) on execution of every bytecode instruction. Upon hitting branch instruction, it duplicates the program state and continues with the execution of both the branches.

In order to implement symbolic execution of target programs, we have modified interpretation of the instructions shown in Table II from the original Rhino. Handling of instructions for stack manipulation, exception handling, and variable scope management remain intact.

For example, an instruction ADD op1 op2 is interpreted as follows.

1) Operands op1 and op2 are popped from stack. The operands may take either symbolic or concrete value.
2) Types of the operands are checked. If both the operands are String, the result of computation is the concatenation of the operands. If they are Numeric, the result is the sum of the operands. Otherwise, values are converted according to ECMAScript language standard, and the result is either concatenation or addition of the obtained values.

As JavaScript is a dynamically typing language, types of operands for Rhino instructions are not known prior to execution. Therefore, types of results also need to be determined at run time. For example, evaluation of instructions v1 = ADD e1:number v:untyped; v2 = ADD v1 "abc" yields, v1 = e1 + v:number; v2 = toString(e1 + v:number) concat "abc". Types of variables v1,v2 are fixed just at run time in a dynamic fashion.

Comparison instructions are followed by branch instructions in Rhino bytecode. SymJS handles compare and branch instruction pairs as in the following. First, it creates Boolean formula corresponding to result of comparison after necessary type conversions. Assuming the created formula is denoted by symbol c, we check if c and its negation $\neg c$ are satisfiable together with current path condition pc. In other words, we check for satisfiablity of $pc \wedge c$ and $pc \wedge \neg c$. If both are satisfiable, we build states s_1, s_2 corresponding to $pc \wedge c$ and $pc \wedge \neg c$ and continue with execution from states s_1 and s_2. If one of them is satisfiable, the state corresponding to the satisfiable one is chosen and execution resumes from that point.

SymJS supports two ways to manage states, which are forked on hitting branches etc. The first method is to store all program state variables including content of heap/stack, as is

Table III. Comparison of Symbolic Executors

Tool	Target Lang.	Sym. VM	Dep./Cache Solving	String Solving
SymJS	JavaScript	Yes	Yes	Yes
KLEE [3]	C	Yes	Yes	No
SAGE [7]	x86 binary	No	Yes	No
Sym JPF [4]	Java	Yes	No	No
Kudzu [14]	JavaScript	No	No	Yes
Jalangi [15]	JavaScript	No	No	Limited

Table IV. Representation of States in Fuzzing after Executing Code on Figure 1 under Path Conditions in Table I

Test No.	Blocks Executed	State Representation
1	0,A	L;L
2	0,B	L;R
3	1,A	R;L;L
4	1,B	R;L;R
5	3,A	R;R;R;L
6	3,B	R;R;R;R

done in [3], [4]. The second method is to remember only which side is taken on branches. This method needs to re-execute the target program from its initial state on backtracking. However, it benefits from its simple implementation and smaller memory footprint. The method is called "Fuzzing" and similar to the technique introduced in [5], [7]. However, our technique is implemented upon our symbolic executor and does not need modification of target code required in the existing tools [14], [15] for JavaScript.

During symbolic execution of programs through fuzzing, states are represented and stored only by which side is taken on branches. The information can be used to re-execute the program from its initial state and explore the state space target may take. States after symbolically executing the target program in Figure 1 with path conditions corresponding to tests 1-6 in Table I, are represented as shown in Table IV during fuzzing. Symbols *L,R* denote left/right branch is taken on a branching instruction.

For each of state representations shown in Table IV, corresponding path condition can be obtained. Table I includes path conditions for the states in Table IV. If it is possible to obtain solutions satisfying the constraints, they can be used as inputs used during testing. Constraints on numbers can be solved by feeding them into SMT solvers. However, SMT solvers cannot handle constraints of strings, which is heavily used in most of JavaScript code. Therefore, we employ constraint solver PASS [17] during test input generation.

PASS can handle constraints over integers, bit-vectors, floating-point numbers, and strings. While previous constraint solvers with support for string constraints used bit-vectors or automata, PASS introduced modeling through parameterized-arrays, which allows for more efficient solving. PASS converts string constraints into quantified expressions. The expressions are solved through an efficient and sound quantifier elimination algorithm. The algorithm speeds up identification of unsatisfiable cases. As the consequence, it can solve complex constraints corresponding to string manipulations within ECMAScript standard. Multiple optimizations are also introduced on incorporating PASS into SymJS. Such optimizations include dependency solving, cache solving and expression simplification to reduce computation within the solver.

```
symjs_assume(arg0.length==16);
```

Figure 3. Use of symjs_assume() to Constrain Length of arg0 to be 16

As the nature of symbolic execution, SymJS may suffer from path explosion on targeting programs with large state space. In order to eliminate program behavior of uninterest, SymJS can make use of `symjs_assume(assumption)` function call, which prunes state space violating the assumption. The code snippet shown in Figure 3 shows an example of constraining length of string `arg0` to be *16*.

C. Symbolic Stubs and Drivers

Symbolic variables are targets of test input generation through symbolic execution. SymJS allows definition of symbolic variables through function calls. The code snippet below shows an example of defining symbolic string variable. `var s = symjs_mk_symbolic_string();` While the example defines a symbolic variable of string type, functions `symjs_mk_symbolic_int()`, `symjs_mk_symbolic_bool()` and `symjs_mk_symbolic_real()` allow definition of symbolic variables with their type being integer, Boolean, and floating-point, respectively. While SymJS allow only string, integer, Boolean, and floating-point numbers to be symbolic, their constraints are retained on assignments/references as fields of more complex objects, allowing generation of tests with values of object fields varied.

In order to determine test inputs for the function `func0()` in Figure 1, additional code fragments are required. First, a symbolic driver shown in Figure 6 is required. The driver declares symbolic variables and passes them to the function as arguments. Stubs to inject dependencies are also required. A symbolic stub in Figure 7 includes a symbolic variable declaration. With the stub, return values of function `Lib.m2()` are included to test inputs obtained by SymJS.

D. Test Execution within Web Browsers

Functions `symjs_mk_symbolic_*()` used to define symbolic variables are interpreted as expressions to define new symbolic variables during test generation. SymJS itself allows for normal concrete test execution with the generated test inputs. During concrete execution, the functions return concrete values contained in test inputs.

SymJS can export test inputs into external test runners based on a test framework Jasmine [11]. The runners contain test playback library, which returns corresponding test input data on `symjs_mk_symbolic_*()` function calls. Figure 4 shows an example of test runner generated. Each of tests contains automatically generated test data in an array structure, and users can easily create new tests through duplication and modification of existing tests.

The runners can be loaded into typical web browser and allow for execution of generated tests with no custom JavaScript interpreter. The runner has an extension to Jasmine, which

```
describe("Test with underscore.string.symjs.js-camelize.js", function() {
    it("should run test 1", function() {
        replayList.init(
[ [ "arg0#0", null ], [ "arg1#1", false ] ]
        );
        var arg0 = symjs_mk_symbolic_string("arg0");
        var arg1 = symjs_mk_symbolic_bool("arg1");
        var retval = camelize(arg0, arg1);
        expect(true).toBe(true);     // default assertion
    });
});
```

Figure 4. Test Runner Code with Test Data

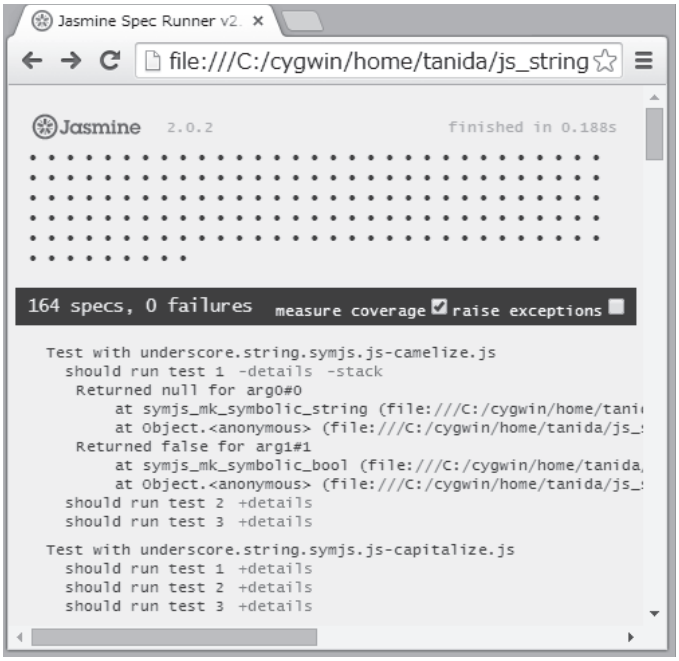


Figure 5. Test Runner View with Test Data and Stacktrace

prints test data and stacktrace on use of test input as shown in Figure 5.

III. Automatic Generation of Symbolic Stubs and Drivers

As explained in Section II-C, symbolic stubs and drivers are required to symbolically execute target functions and obtain test inputs. Symbolic stubs that return symbolic variables are used to generate return values of functions, which are called from functions under test. Symbolic drivers are needed to vary arguments passed to functions tested.

While it is possible to employ manually implemented symbolic stubs and drivers, additional cost is required for implementation. Therefore, it is desirable to have symbolic stubs and drivers automatically generated. Hence, we have decided to generate symbolic stubs and drivers in an automatic manner, and use them for test generation and execution.

A. Strategy for Generating Symbolic Stubs and Drivers

Our symbolic stub generation technique produces stub for functions and classes specified. Our driver generation technique emits code that invokes program under test.

As for stub generation, we have decided to generate functions, which just create and return objects according to type of return value expected by caller. The following is the mapping between expected type and returned object:

- String, integer, Boolean and floating-point numbers which SymJS can handle as symbolic
  (Hereafter referred to as SymJS primitives):
  Newly defined symbolic variable of the corresponding type.
- Other classes:
  Newly instantiated object of the expected type. If the class is targeted for stub generation, newly instantiated stub object is returned.
- Void: Nothing is returned.

In order to create stubs for classes, stubs for constructors also need to be generated. Here, we generate empty constructors, which result in all stateless objects. Our approach assumes there is no direct access to fields of stub classes, and does not generate stubs for fields.

We have to note even in case type of return value from a stub is a non-SymJS primitive, we may get multiple test inputs on invocation on the stub. That is the case if returned objects contain functions that return symbolic variables. The situation happens if the non-SymJS primitive class contain functions that return objects of SymJS primitive class, and the non-SymJS primitive class is stubbed. Therefore, it is possible to obtain more than one set of test inputs by calling functions returning non-SymJS primitive.

Symbolic drivers generated with our technique have the following functionality:

- If the function to be tested is not static and needs an object instance to be executed, instantiate an object of the corresponding class and call the function
- If the function is a static one, just call the function

As arguments passed to the function, drivers give the following objects according to the expected types:

```
var s = symjs_mk_symbolic_string("arg0");
var a = symjs_mk_symbolic_float("arg1");
func0(s,a);
```

Figure 6. Symbolic Driver to Execute Code in Figure 1

```
Lib.m2 = function() {
  return symjs_mk_symbolic_float();
};
```

Figure 7. Symbolic Stub Providing Lib.m2() Used in Figure 1

```
/** @return {Number} m2 value */
Lib.m2 = function() { ... };
```

Figure 8. Function Definition with an Annotation to Automatically Generate Symbolic Stub in Figure 7

```
/** @return {tx.Data} data */
tx.Ui.getValue = function(){ ... };
```

⇓

```
tx.Ui.getValue = function(){
    return new tx.Data();
};
```

Figure 9. Function with an Annotation Returning non-SymJS Primitive and Generated Symbolic Stub

```
/** @param {String} s
  * @param {Number} a */
function func0(s,a) { ... }
```

Figure 10. Annotations for Function under Test to Automatically Generate Symbolic Driver in Figure 6

- SymJS primitives:
  Newly defined symbolic variable of corresponding type.
- Other classes:
  Newly instantiated object of the expected type. If the class is targeted for stub generation, newly instantiated stub object is passed.

The manner to choose arguments is similar to the one resolves what to return in symbolic stubs.

B. Generating Symbolic Stubs and Drivers from Annotations

Symbolic stub/driver generation strategy proposed in Section III-A requires type information from target code. Types of return values expected by caller are required for stub generation. Types of arguments passed to functions under test are required to generate drivers.

However, JavaScript is a dynamically typing language, which makes it difficult to determine type of return values and arguments prior to run time. On the contrary, many JavaScript programs have some expectations in types of return values and arguments, which are often defined in Application Programming Interface (API) etc. Further, there is a way to express type information for JavaScript code in a machine readable manner, which is JSDoc-style annotation. Therefore, we have decided to obtain type information from annotations in JSDoc3 [20] convention, and generate symbolic drivers and stubs.

Symbolic stubs are generated from original source code of functions to generate stubs for. Functions need to contain annotations, which provide type information on return values of functions. Symbolic stub for a function can be generated if the type of its return values is obtained from annotations.

JSDoc3 allows for declaration of return value type, mainly through `@return` annotations. In order to generate symbolic stub for function `Lib.m2()` used in code snippet on Figure 1, an annotation like the one shown in Figure 8 is required. If such annotation is attached to original source code of the function, it is possible to figure out type of return values. From the obtained type of return values, the symbolic stub in Figure 7 can be generated in a fully automatic manner. The example demonstrates generation of symbolic stub for a function returning a SymJS primitive. An example of generating symbolic stub for a function that returns a non-SymJS primitive is shown in Figure 9.

Symbolic drivers are generated from source code of functions to be tested. Source code need to contain annotations expressing type of arguments passed to the function, in order to automatically generate symbolic driver to invoke the function.

Types of parameters passed to functions are often given through `@param` annotation for JSDoc3. Symbolic driver for the function `func0()` can be generated from the annotations in Figure 10, attached to the function. The annotations give types of parameters for the function, allowing generation of the symbolic driver in Figure 6.

The proposed technique for automatic generation of symbolic stub and drivers is implemented as plugins for JSDoc3. JSDoc3 allows implementation of custom plugins, and they may contain hooks to be invoked on finding classes or functions. Within the hooks, it is possible to obtain types for return values and parameters. The developed plugins automatically generate symbolic stubs and drivers for classes and functions found in input source code.

While we have proposed a technique to automatically generate symbolic stubs and drivers based on type information obtained from annotations, it is also possible to use type information from other sources. Such sources of type information include API specification documents.

IV. Evaluation

A. Experimental Setup

In order to confirm that our proposed technique can automatically generate and execute unit tests achieving high code coverage, we have performed experiments using two JavaScript programs with their statistics shown in Table V.

The first subject (INDUSTRIAL) corresponds to the client part of web application implemented upon our in-house framework for web application implementation. Within the target

Table V. Statistics on the Target Program

Name	INDUSTRIAL	UNDERSCORE.STRING
#Statement	123	427
#Public Function	22	57

program, calls to API not defined in ECMAScript standard are wrapped in our framework. As the consequence, it contains only calls to standard API or our framework. We have to note common API to manipulate HTML Document Object Model (DOM) or to communicate with servers are not part of ECMAScript standard and not used directly in the program.

The second subject (UNDERSCORE.STRING) is a free and open-source string manipulation library Underscore.string [21]. It provides many useful string operations, which are not standardized in JavaScript programming language. The library has no functionality involving HTML DOM manipulation or communication, and implemented using only functionality defined in ECMAScript standard. It can be employed on server-side Node.js platform as well as web browsers on clients. We have manually annotated source code of the target to provide argument type information required for automatic driver generation.

All experiments are performed on a workstation with Intel Xeon CPU E3-1245 V2@3.40GHz and 16GB RAM.

B. Generation of Symbolic Stubs and Drivers

In order to perform automatic generation of test input proposed in Section II, we have generated symbolic stubs and drivers with the technique explained in Section III.

Symbolic stubs to target INDUSTRIAL are generated from source code of the framework used to implement the application. Source code has annotations meeting JSDoc3 standard, which allow for retrieval of types for return values of functions. Stubs are successfully generated for all the classes and functions defined in the framework. UNDERSCORE.STRING that depends only on functionality provided by ECMAScript standard required no stub to be generated. As the program is implemented only upon API defined in ECMAScript language standard and the framework, all the stubs required for symbolic execution of the program are ready at this stage.

Symbolic drivers are generated from source code of the program under test. INDUSTRIAL had JSDoc3-style annotations as is. Manually annotated source code of UNDERSCORE.STRING is used to extract type information for function arguments within the subject. Types of arguments in UNDERSCORE.STRING could be found in its document, and the annotation process was straightforward. Drivers for all functions within the two targets are generated successfully.

C. Test Input Generation

All functions within the target programs are symbolically executed using the automatically generated drivers and stubs. Table VI contains statistics on the generated tests.

In the first trial, the subject programs are symbolically executed with no special configuration. While all functions in INDUSTRIAL are processed within *1* second, analysis of *2* functions in UNDERSCORE.STRING did not finish within timeout of *1* minute. In order to process the functions within reasonable time, we have introduced the following constraints during symbolic execution of UNDERSCORE.STRING.

- `count(string,substring)`, which counts number of `substring` occurrences in `string`, resulted in timeout assuming arbitrary string as `string` and `substring`. This is due to non-terminating loop and resulting large number of forked states during symbolic execution. we have limited maximum number of branches a state may go through to `7` from its default configuration of `20`.
- `words(string,delimiter)`, which counts number of words within `string` separated by `delimiter` also resulted in timeout, assuming arbitrary string as the two parameters. We have constrained length of two arguments to equal to *16* and *1*, respectively. Constraints can be introduced through insertion of `symjs_assume()` calls to the driver.

After introducing the constraints, all functions within UNDERSCORE.STRING are processed within *2* seconds.

D. Test Execution

Test inputs for all functions in INDUSTRIAL and UNDERSCORE.STRING are automatically exported to test runners based on Jasmine test framework. Code coverage during testing is measured with Blanket.js [22], and line coverage of *92.7%* and *76.0%* on average is obtained for INDUSTRIAL and UNDERSCORE.STRING, respectively.

Figure 11 shows distribution of statement coverage for functions in the benchmarks. The result shows our technique can generate unit test input with high coverage. For instance, 100% statement coverage is achieved with more than 60% of the functions. However, some of the benchmarks are not fully covered due to limitations in symbolic execution or stub/driver generation. In the sequel, we discuss the source of failure to cover some statements.

E. Code Not Covered in the Experiments

While the experimental results show that the proposed method can generate test input achieving high code coverage, 100% coverage is not reached, implying some portion of the target program is not exercised. Automatically generated test runner code shown in Figure 4 allows for manual modification and insertion of cases in order to test such code. However, additional labor is required and it is desirable to have such code automatically covered. The followings are the classes of code not executed, and possible enhancements to our methodology, which allows for coverage of missed code.

1) Code Exercised on Matches to Regular Expressions

JavaScript features regular expression library within its core ECMAScript language standard. The functionality is very useful to perform string manipulations and heavily used in UNDERSCORE.STRING. However, SymJS cannot obtain and manage path conditions corresponding to matches and unmatches on some regular expressions. The limitation results

Table VI. Statistics on the Tests Generated

Target	INDUSTRIAL	UNDERSCORE.STRING (w/o constraints)	UNDERSCORE.STRING (w/ constraints)
Functions with generated tests	22	55	57
#Test per function(max./avg./min.)	27/2.6/1	201/24.4/1	48/6.4/1
Max. test generation time per function (sec.)	< 1	> 60 (timed out)	< 2
Statement coverage(%)	92.7	73.5	76.0

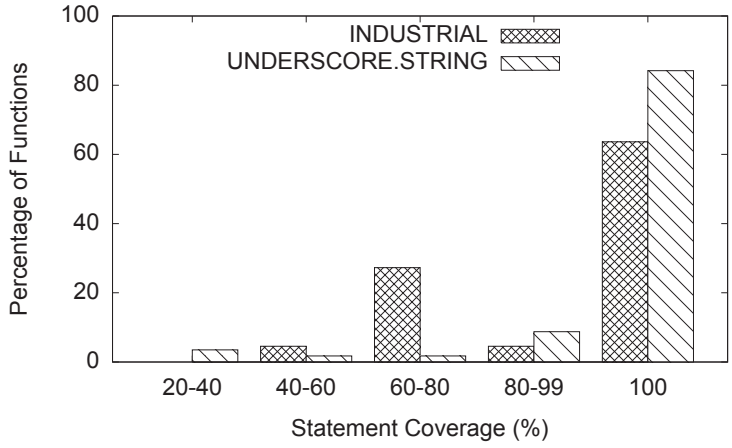


Figure 11. Distribution of Statement Coverage for Functions in INDUSTRIAL and UNDERSCORE.STRING

```
// _fmt is a symbolic String variable
// created from arguments to the function
if ((match = /^[^\x25]+/.exec(_fmt))
    !== null) {
    parse_tree.push(match[0]);
}
```

Figure 12. Code Exercised on Matches to Regular Expressions from sprintf() Function in UNDERSCORE.STRING

```
/**
 ...
 * @param {Number} position
 */
function endsWith(str, ends, position) {
  ...
  if (typeof position == 'undefined') {
    position = str.length - ends.length;
  }...
}
```

Figure 13. Code Handling Objects of Unexpected Type from endsWith() Function in UNDERSCORE.STRING

in failure to cover code to be exercised on matches of input string to such regular expressions.

11 functions in UNDERSCORE.STRING are not fully covered due to this limitation in handling regular expressions. Figure 12 shows such code fragment found in `sprintf()` function from UNDERSCORE.STRING. The restriction results in 57/75 statements within the large function missed, downpressing total coverage achieved with UNDERSCORE.STRING.

We are planning to enhance string constraint solver PASS, in order to model and handle larger class of regular expressions.

2) Code Handling Objects of Unexpected Type

As JavaScript is a dynamically typing language, objects of unexpected type might be returned from functions. In order to handle such scenario, the target programs contained type checking and subsequent branching code. However, symbolic stubs generated through our technique, always return an object of type specified in source code annotation. Such stubs fail to cover code portions handling objects of type different from annotations.

1 function in INDUSTRIAL and *2* functions in UNDERSCORE.STRING contain such code, and full coverage is not achieved. Figure 13 shows the corresponding code fragment from `endsWith()` function in UNDERSCORE.STRING.

Code handling objects of unexpected type can be exercised by making use of multiple symbolic stubs/drivers, which return/pass objects of different type. Currently, we support `@return` and `@param` annotations each specifying single type. However, JSDoc3 includes support for annotations with multiple possible types of arguments and return values. Our symbolic driver and stub generator can be extended easily to support such annotations.

3) Code with No Premise on Object Type

INDUSTRIAL also contained *1* function, which determines type of objects at run time and process them accordingly.

However, our technique cannot cover such procedures. From functions with types of their return values unknown, we generate stubs that return the default JavaScript "Object". As the consequence, code interacting with objects of custom class is uncovered.

Object types a variable may take can be extracted by means of static analysis or random tests. Our symbolic execution technique can be employed to create test input variation within the obtained type.

4) Code Iterating through Entries in Hash or Array

1 function within INDUSTRIAL had loop iterating through members in objects returned from a symbolic stub. Such control structure is often observed in JavaScript code, in order to make use of a plain "Object" as a hash table. However, automatically generated symbolic stub returns newly created "Object" with no members, and loop body is missed in the experiment. Code that inspects content of arrays from symbolic stubs is also expected to missed, as symbolic stubs generated by our tool return newly created empty arrays.

Loop bodies in such functions can be exercised by stubs that returns "Object" or array with one or more members contained.

5) Catch Blocks Handling Exceptions

1 function from INDUSTRIAL contained catch blocks for exceptions thrown from the framework used in the program. However, after replacing the framework with the automatically generated symbolic stubs, exceptions are never thrown and catch blocks are not exercised.

In order to cover catch blocks in target, we have to generate symbolic stubs that throws exception during execution, in addition to those do not throw exceptions.

V. DISCUSSION

A. Completeness

Our symbolic execution technique depends on modeling of computation performed, including complex one such as string manipulations. While we have employed constraint solver PASS, which is capable of handling complex string constraints, some constraints such as the one corresponding to code with regular expressions could not be handled.

Another limitation comes from automatically generated symbolic stubs/drivers from our technique. Type annotations used as input of our stub/driver generation technique, do not provide enough information to model environments where target software is executed. For instance, target software might be fed objects of unexpected type, or thrown exceptions. Our symbolic stub/driver generation technique does not take such situations into account.

However, compared to existing industrial practice of testing based on manually written tests, our technique can test behavior of wider scope in an automatic fashion.

B. Scope

Like other automated test input generation techniques based on symbolic execution, we do not automatically generate assertions to check target application behavior conformance with the obtained test inputs. In other words, users need to provide assertions/invariants to ensure target code is functioning as expected.

However, invariants and many assertions can be shared between multiple test cases, and costs for writing them is much smaller compared to those required to write tests from scratch. Further, our Jasmine-based test runner allows for insertion of global invariants, as well as to assertions specific to test cases.

C. Automation Level

Our test generation technique may require some user inputs on targeting complex applications such as UNDERSCORE.STRING. In the experiment, maximum number of branches and constraint to string length are required to end test generation for some functions. However, the number of parameters that needed to be adjusted are quite small. In addition, more than 95% of functions in the subject could be handled with the default configuration of the symbolic executor.

D. Correctness

Symbolic stubs generated with our technique always return newly defined symbolic variable. Such behavior may result in over-approximation of real system behavior before introducing stubs. For example, an expression containing multiple calls to a single function `getValue()!=getValue()` is unlikely to be satisfied, assuming target variable of the getter functions is not accessed from other execution contexts between two function calls. However, as the stubbed `getValue()` function returns newly defined symbolic variable on every call, it is possible to generate tests that make the expression true.

Combination of the generated test input and stub allows for reproduction of behavior that makes the expression true during test execution. However, the behavior might be quite different and hard to reproduce in real system under development. In that case, stubs need to be manually modified or developed, in order to mock behavior of environment in which software under test is executed.

E. Scalability

Our case studies confirm that our technique can be applied to real-world JavaScript code used in field. While we need to adjust some parameters used in test generation, we were able to target all functions in the benchmarks within *2* seconds.

However, we need to perform experiments on applications with their size varied. In order to target software of larger scale, test generation techniques such as DART [23] would be required, in addition to pure symbolic execution used in current SymJS.

F. Threats to Validity

Issues related to the external validity of our evaluation are handled in the discussions above. The internal validity of our experiments may depend on software tools used. We have minimized the chance by writing tests for the toolchain developed, which are completely different from the benchmarks used in the evaluation.

VI. Conclusions and Future Work

A. Conclusions

We proposed a technique to automatically generate unit test input data for JavaScript code. The technique makes use of a symbolic executor SymJS, in order to achieve high code coverage during testing. The technique is a two-phase approach, consisting of the following fully-automatic steps:

1) Symbolic stub/driver generation based on type information obtained from annotations
2) Test input generation through symbolic execution of target code

The experiments were conducted targeting client part of proprietary web application and open-source string manipulation library. Our technique generated tests that achieve line coverage higher than 75%, and more than 60% of functions in the subject are fully covered with the generated tests. The results show the technique can automate generation and execution of high-coverage unit tests for large portion of JavaScript code in the field.

B. Future Work

Future work includes more verification trials with variety of target programs. While we have performed experiments with programs of relatively small size, experiments on larger targets are also required.

In order to exercise target code missed in the experiment automatically, constraint solver PASS and symbolic stub/driver generator need to be improved. However, our methodology allows for manual modification of generated tests to cover such code, which is found in less 40% of the subject in the evaluation.

In the experiment, we have targeted JavaScript code with HTML DOM API encapsulated in our custom framework (INDUSTRIAL) and code that involves API defined in ECMAScript standard associated only (UNDERSCORE.STRING). As the consequence, symbolic stubs required for test generation and execution in the experiment were only those corresponding to our custom framework. However, in order to target JavaScript code, which has interactions with server-side API such the one in Node.js or client-side API for HTML DOM manipulations, symbolic stubs for corresponding APIs need to be developed. To target mobile applications, it is required to prepare symbolic stubs for frameworks used in their implementation. Development support techniques for new symbolic stubs are necessary, in order to support larger set of platforms with JavaScript code deployed with reduced effort.

References

[1] H. Tanida, G. Li, M. Prasad, and T. Uehara, "Automatic Unit Test Generation and Execution for JavaScript Program through Symbolic Execution," in Proceedings of the Ninth International Conference on Software Engineering Advances, 2014, pp. 259–265.

[2] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "EXE: Automatically Generating Inputs of Death," in Proceedings of the 13th ACM Conference on Computer and Communications Security, 2006, pp. 322–335.

[3] C. Cadar, D. Dunbar, and D. Engler, "KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs," in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 209–224.

[4] C. S. Păsăreanu and N. Rungta, "Symbolic PathFinder: Symbolic Execution of Java Bytecode," in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, 2010, pp. 179–180.

[5] K. Sen, D. Marinov, and G. Agha, "CUTE: A Concolic Unit Testing Engine for C," in Proceedings of the 10th European Software Engineering Conference, 2005, pp. 263–272.

[6] N. Tillmann and J. De Halleux, "Pex: White Box Test Generation for .NET," in Proceedings of the 2nd International Conference on Tests and Proofs, ser. TAP'08, 2008, pp. 134–153.

[7] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox Fuzzing for Security Testing," Queue, 2012, pp. 20:20–20:27.

[8] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: Introduction and Applications," Commun. ACM, vol. 54, no. 9, 2011, pp. 69–77.

[9] "Node.js," https://nodejs.org/, [Online; accessed 2015.05.28].

[10] "PhoneGap — Home," http://phonegap.com/, [Online; accessed 2015.05.28].

[11] "Jasmine: Behavior-Driven JavaScript," http://jasmine.github.io/, [Online; accessed 2015.05.28].

[12] "QUnit: A JavaScript Unit Testing framework," http://qunitjs.com/, [Online; accessed 2015.05.28].

[13] "Mocha - the fun, simple, flexible JavaScript test framework," http://mochajs.org/, [Online; accessed 2015.05.28].

[14] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A Symbolic Execution Framework for JavaScript," in Proceedings of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 513–528.

[15] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs, "Jalangi: A selective record-replay and dynamic analysis framework for javascript," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 488–498.

[16] G. Li, E. Andreasen, and I. Ghosh, "SymJS: Automatic Symbolic Testing of JavaScript Web Applications," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 449–459.

[17] G. Li and I. Ghosh, "PASS: String Solving with Parameterized Array and Interval Automaton," in Proceedings of Haifa Verification Conference, 2013, pp. 15–31.

[18] ECMA International, Standard ECMA-262 - ECMAScript Language Specification, 5th ed., June 2011. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[19] "Rhino," https://developer.mozilla.org/en-US/docs/Rhino, [Online; accessed 2015.05.28].

[20] "Use JSDoc," http://usejsdoc.org/index.html, [Online; accessed 2015.05.28].

[21] "underscore.string," https://epeli.github.io/underscore.string/, [Online; accessed 2015.05.28].

[22] "Blanket.js —Seamless javascript code coverage," http://blanketjs.org/, [Online; accessed 2015.05.28].

[23] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," in Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2005, pp. 213–223.

Quality-Oriented Requirements Engineering of RESTful Web Service for Systemic Consenting

Michael Gebhart, Pascal Giessler
iteratec GmbH
Stuttgart, Germany
michael.gebhart@iteratec.de,
pascal.giessler@iteratec.de

Pascal Burkhardt, Sebastian Abeck
Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
pascal.burkhardt@student.kit.edu,
abeck@kit.edu

***Abstract*—Making decisions is a typical and recurring challenge in a society as humans often have different opinions concerning a certain issue. Consensuses have to be found that satisfy all participants. To support the finding of consensuses, at the Karlsruhe Institute of Technology a new software service is developed, the Participation Service, to support the systemic consenting. This service is expected to be part of the already existing service-oriented campus system of the university that supports students in their daily life. The Participation Service is expected to be developed in an agile manner. Furthermore, as the entire architecture is based on the Representational State Transfer paradigm, also the new service is expected to be RESTful. One of the key success factors of such projects is the gathering of requirements as the software bases on them. In agile projects, scenarios are an appropriate way to describe a system from the user's point of view. However, it is not obvious how to specify the requirements so that they are of high quality. This article presents an enhancement of scenario-based requirements engineering techniques, so that the resulting requirements fulfill the quality characteristics of the international standard ISO/IEC/IEEE 29148. The requirements engineering technique has been created for the development of RESTful web services. For that reason, this article demonstrates its application by means of the Participation Service. Functional and non-functional requirements are elicited and constraints that emerged from the existing RESTful service-oriented architecture are considered.**

Keywords: requirements engineering; agile; scenario; rest; service; participation; iso 29148

I. INTRODUCTION

This article is an extended version of [1]. It describes the requirements engineering approach that has been applied for the Participation Service, a web service for systemic consenting more in detail. Furthermore, compared to the original work, it is shown that the approach is not necessarily limited to RESTful web services as REST is only a constraint in the methodology. Nevertheless, the focus is still on web services in a service-oriented architecture. The general applicability on all kind of software systems is possible, but not yet proven. This kind of applicability should be considered in future research work.

Decision-making is always a typical and recurring challenge in a society. When having a certain issue, stakeholders and participants have different opinions. They defend their points of view and try to convince the others of their personal opinion. To make a decision, consensuses have to be found that satisfy all stakeholders and participants.

At the Karlsruhe Institute of Technology (KIT) a new software service, the Participation Service, is expected to be developed that supports the finding of consensus. The Participation Service is based on the idea of systemic consenting. This approach describes how to find a compromise or consensus that is near to an optimal consensus for the entire group and all stakeholders and participants. For that purpose, possible solutions are scored with points. However, compared to usual decision-making processes, the solutions are not scored with agreement points but with refusing points. This means, after describing the issue and collecting possible solutions, the one solution is selected that has the fewest refusing points. This solution represents the one with minimum resistance.

The Participation Service is expected to be part of the already existing service-oriented campus system of the university. The so-called SmartCampus is a system that provides functionality for students to support their daily life. For example, today the SmartCampus offers functionality to find free workplaces or to determine the route to a certain destination, such as the library of the university. As the services of the SmartCampus are expected to be used by several different devices, such as notebooks, smartphones and tablets, the software services are developed as web services based on the Representational State Transfer (REST) paradigm [2] as lightweight alternative to technologies, such as SOAP over Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), and Web Services Description Language (WSDL). The RESTful web services are invoked by a web application that is responsive and can be therefore used on all the required devices. Furthermore, the service is developed in an agile manner to rapidly receive feedback about its usability.

For successful software projects, one key success factor is the requirements engineering with its underlying process and methodology [3][4]. In this phase, the functional and

non-functional requirements are gathered and described. They represent the basis for the entire software project. In agile projects, the usage of scenario has evolved as an appropriate way to describe the requirements. Scenarios represent the requirements from a user's point of view. As the entire software project bases on the requirements, their high quality is very important. For that reason, the IEEE has created a set of quality characteristics for requirements. They are summarized in the IEEE recommended practice for software requirements specifications IEEE Std 830-1998 [5] and its successor, the ISO/IEC/IEEE 29148 [6]. However, existing scenario-based requirements engineering methodologies do not consider these quality characteristics explicitly.

This article enhances existing requirements engineering methodologies for agile projects in a way that quality characteristics of the international standard ISO/IEC/IEEE 29148 [6] are considered. For that purpose, in a first step, existing methodologies are analyzed and described. In a next step, the most appropriate methodology is reused and adapted where necessary. In this phase, these parts of other methodologies that support the achievement of certain quality characteristics of the international standard are reused and combined with the chosen methodology. As result, a methodology is created that combines the best parts of all analyzed methodologies.

To illustrate the resulting methodology, the Participation Service for the SmartCampus as a real-world project is considered. Its requirements are gathered and described using the elaborated methodology. Based on this approach, in a first step, the stakeholders are identified. Afterwards, the goals of the Participation Service are elicited and prioritized. In the last step, the functional and non-functional requirements are formalized and it is shown that they fulfill the quality characteristics of ISO/IEC/IEEE 29148.

The article is structured as follows: Section II examines existing work in the context of requirements engineering methodologies and quality characteristics for requirements. Section III introduces the Participation Service as exemplarily scenario. In this context, the idea behind the service is described in detail. Our quality-oriented requirements engineering methodology is presented in Section IV. Section V concludes this article and introduces future research work in the context of a quality-oriented development of RESTful web services.

II. BACKGROUND

This section analyzes existing approaches in the context of requirements engineering methodologies that identify the goals of stakeholders and writes them down in a precise way so that they can be used in the following development phases [7].

In IEEE Std 830-1998 [5], the IEEE offers an official recommended practice for software requirements specifications, which was replaced by the new international standard ISO/IEC/IEEE 29148 [6]. Based on them, quality characteristics for high quality requirements can be derived. Furthermore, the new standard provides language criteria for writing textual requirements and requirements attributes to support requirement analysis. It also provides guidance for applying requirements-related processes. These concepts will be used to analyze existing scenario-based requirements engineering methodologies and to design the one introduced in this article.

Sharp et al. [8] present a domain-independent approach for identification of the stakeholders based on four determined groups of so-called baseline stakeholders. They can be further refined into three different groups based on their role. This approach will be used to identify the stakeholders in this article. However, in large projects the resulting network of stakeholders can be huge.

For that reason, Ackermann et al. [9] describe a method with a matrix in which the stakeholders were arranged by their importance and their influence on the project. This method can be used to prioritize the discovered stakeholders for the project.

There are different requirement types, which have to be taken into account when eliciting requirements for a software product. Glinz [10] provides a concern-based taxonomy of requirements, which consists of functional requirements, non-functional requirements, and constraints. These types will be reflected in the introduced requirements engineering methodology, however with one difference: The performance will not be considered as a separate entity since it is already an ingredient of ISO/IEC 25010:2011 [11].

For eliciting functional requirements, Rolland et al. [12] present a goal modeling approach by using scenarios. A goal represents something that the stakeholders want to achieve in the future, while a scenario represents the required interactions between two actors to achieve the corresponding goal. Once a scenario has been composed, it is investigated to addict more goals. This approach can be aligned with ISO/IEC/IEEE 29148 [6], which is why it will be reused in this article.

However, there are two issues: 1) Goals cannot be regarded separately because they could be composed of existing goals and 2) the recursive process is repeated until no more subgoals can be derived, but this can lead to a big bunch of subgoals. A solution for 1) is a repository of already analyzed goals, which can be reused by reference. The determination of a threshold in 2) is difficult, because it cannot be set easily by metrics. So the requirements engineer has to decide on its own when the abstraction meets its expectations. For this purpose, some conditions had to be found, which support the decision-making. Furthermore, it is not obvious how to achieve the initial goals.

At this point, Bruegge and Dutoit [13] introduce some interview questions that can be used for identification of the initial goals. Furthermore, elicitation techniques can be found in [3]. To support agile software engineering, the discovered goals have to be arranged by importance to select the goals with the highest rank similar to iteration.

For that reason, the approach by Karlsson and Ryan [14] will be applied, which uses pairwise comparisons in consideration of cost and value. But, for many goals, this approach will rapidly become impracticable as the number of comparisons increases significantly. For that reason and the statement "Keep the prioritization as simple as possible to

help you make the necessary development choices" by Wiegers [15], a simple classification approach with three different scales based on IEEE Std 830-1998 [5] is best suited for the initial prioritization.

When writing scenarios, the quality characteristics by [6] have to be considered. Glinz [16] presents an approach, which respects the quality characteristics by the old recommendation IEEE Std 830-1998 [5]. His findings will be used to improve the quality of requirements.

Also, Terzakis [17] presents techniques for writing higher quality requirements by providing an overview of requirements and pitfalls by using the natural language for their description. Based on this, the quality of requirements will be improved even further.

In [11], the ISO provides a quality model comprising quality characteristics that are further decomposed into sub-characteristics. This model will be used for determining the quality aspects of a software product.

For eliciting non-functional requirements, the approach by Ozkaya et al. [18] will be used. Due to the fact that statements like "The system shall be maintainable" are imprecise and not very helpful, this approach is using so-called quality attribute scenarios. Based on these, the corresponding quality characteristic of ISO 25010 [11] can be derived. However, for many quality characteristics it can be very time-consuming.

To reduce the effort, the decision-making approach by Saaty [19] will be applied by using pairwise comparison of the quality characteristics in ISO/IEC 25010:2011 [11] with regard to their importance for the product strategy.

With the provided constraints of the architectural style REST in [1], the last requirement type according to the taxonomy in [10] will be considered.

III. SCENARIO

To illustrate the requirements engineering approach, the SmartCampus System at KIT is to be enhanced by a new service, the Participation Service. The SmartCampus system is a service-oriented system to support professors, students, and other KIT members in their daily life. For example, the SmartCampus system already provides services to determine the route to a certain room or to find free workplaces.

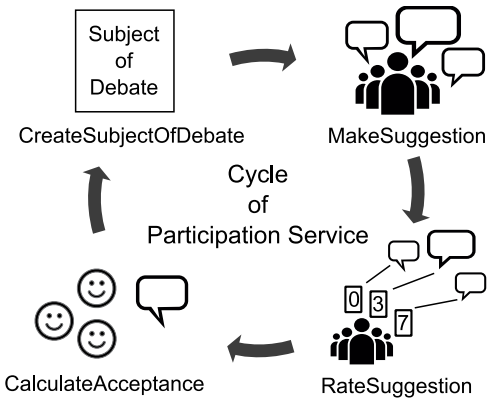


Figure 1. Systemic consenting process.

The services of the SmartCampus can be used by means of web applications that can be also used on mobile devices, such as smartphones and tables. For that reason, the web applications are developed with a responsive layout using modern and standardized web technologies, such as Hypertext Markup Language (HTML) 5.

The Participation Service is designed to support the process of decision-making between professors, students, and other KIT members according to the principle of systemic consenting. In the first phase, participants can create and describe their own subjects of debate and share them to a group of participants. In the second phase, the participants rate suggestions by expressing their dislike instead of their like as usually expected. They are able to do that in the form of refusing points from zero to ten. Refusing points indicate how much a participant dislikes a possible suggestion. Thus, rating a suggestion with zero refusing points means that the participant totally agrees with the suggestion. Rating a suggestion with ten refusing points means that the participant rejects the suggestion. The suggestion with the fewest amount of refusing points represents the one with the highest acceptance of all participants. This suggestion has minimum resistance and is the consensus of the group. Fig. 1 illustrates the described process. For example, the Participation Service can be used for determining new lecture contents in collaboration with students in the context of the Research Group Cooperation & Management (C&M).

For illustration of our scenario-based requirements engineering technique, the simple goal "Rate a suggestion" of the Participation Service was chosen: A participant requests the website of the Participation Service and gets to see a login screen. After he logged in correctly, he gets a list of subjects of debate. He selects a subject of debate, which he is interested in. He sees a description of the subject and a list of suggestions sorted descending by acceptance. Once reading all suggestions, the participant rates each suggestion with refusing points from zero to ten to express his dislike against the suggestion. The Participation Service updates the acceptance of each suggestion and rearranges them.

IV. QUALITY-ORIENTED REQUIREMENTS ENGINEERING OF RESTFUL WEB SERVICE FOR SYSTEMTIC CONSENTING

In this section, our requirements engineering methodology is introduced. This represents our proposed solution for gathering requirements that verifiably fulfill quality attributes introduced in ISO/IEC/IEEE 29148 [6]. This can be proven to the customer. First, the quality characteristics of the standards IEEE Std 830-1998 [5] and ISO/IEC/IEEE 29148 [6] are presented. Next, the stakeholders are identified followed by an elicitation of their goals. With the prioritization of the goals, they are selected for the iteration. Afterwards, the functional and non-functional requirements are discovered and documented according to the derived quality characteristics of [6] and the provided taxonomy by Glinz [10]. Finally, the elicited requirements for iteration were verified according to specific quality characteristics in [6]. The entire requirements engineering methodology is shown in Fig. 2.

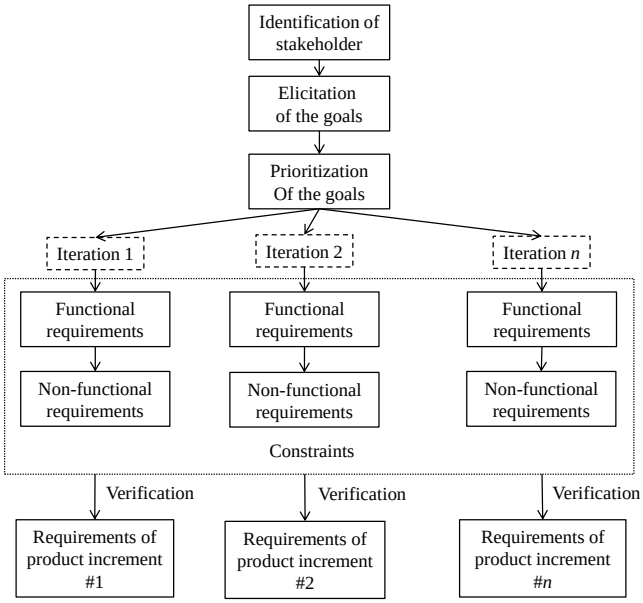


Figure 2. Requirements engineering methodology for agile development of RESTful web services.

A. Quality Characteristics for Requirements

According to IEEE Std 830-1998 [5], the requirements quality focuses on correctness, unambiguousness, completeness, consistence, prioritization, verifiability, modifiability, and traceability. The IEEE Std 830-1998 [5] was replaced by the international standard ISO/IEC/IEEE 29148 [6], which introduces feasibility, necessity, free of implementation, and singularity as new characteristics for requirements while removing prioritization, correctness and modifiability. Furthermore, the new standard distinguishes between individual and a set of requirements. According to them, a set of requirements shall be complete, consistent, affordable, and bounded. The full set of quality characteristics with its definition is shown in Tables I and II [6].

TABLE I. QUALITY CHARACTERISTICS FOR INDIVIDUAL REQUIREMENTS

Quality Characteristic	Definition
Necessary	"The requirement defines an essential capability, characteristic, constraint, and/or quality factor. If it is removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities of the product or process…" [6]
Implementation free	"The requirement, while addressing what is necessary and sufficient in the system, avoids placing unnecessary constraints on the architectural design…" [6]
Unambiguous	"The requirement is stated in such a way so that it can be interpreted in only one way. The requirement is stated simply and is easy to understand." [6]
Consistent	"The requirement is free of conflicts with other requirements." [6]
Complete	"The stated requirement needs no further amplification because it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder's need." [6]
Singular	"The requirement statement includes only one requirement with no use of conjunctions." [6]
Feasible	"The requirement is technically achievable, does not require major technology advances, and fits within system constraints (e.g., cost, schedule, technical, legal, regulatory) with acceptable risk." [6]
Traceable	"The requirement is upwards traceable to specific documented stakeholder statement(s) of need… The requirement is also downwards traceable to the specific requirements in the lower tier requirements specification or…" [6]
Verifiable	"The requirement has the means to prove that the system satisfies the specified requirement. Evidence may be collected that proves that the system can satisfy the specified requirement…" [6]

In [1], we took the assumption that the full set of quality characteristics can be fulfilled by ensuring the individual ones. But, this is not true for the full set of quality characteristics since a complete requirement does not provide information about the completeness of a set of requirements.

TABLE II. QUALITY CHARACTERISTICS FOR A SET OF REQUIREMENTS

Quality Characteristic	Definition
Complete	"The set of requirements needs no further amplification because it contains everything pertinent to the definition of the system or system element being specified." [6]
Consistent	"The set of requirements does not have individual requirements which are contradictory. Requirements are not duplicated. The same term is used for the same item in all requirements." [6]
Affordable	"The complete set of requirements can be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory)." [6]
Bounded	"The set of requirements maintains the identified scope for the intended solution without increasing beyond what is needed to satisfy user needs." [6]

Due to that, we formalized the quality characteristics in Table II in a way that it can be applied on a set of requirements for easier quality control at the end of a requirements engineering phase. The formalization for each quality characteristic is shown in Equations (1)-(4), while Table III will give the explanation of the used elements. The necessary information for the interpretation of the results will be given in Table IV.

$$\mathrm{COM}(\mathrm{R_d}) = \frac{\mathrm{R_d} \cap \mathrm{R_s}}{\mathrm{R_s}} \text{ if } |\mathrm{R_s}| > 0 \text{ else } 1 \quad (1)$$

$$CON_1(\mathrm{R_d}) = 1 - \frac{|R(R_d)|}{|\mathrm{R_d}|} \text{ if } |\mathrm{R}_d| > 0 \text{ else } 1 \quad (2)$$

$$CON_2(\mathrm{R_d}) = 1 - \frac{|C(R_d)|}{|\mathrm{R_d}|} \text{ if } |\mathrm{R}_d| > 0 \text{ else } 1$$

$$CON_3(\mathrm{R_d}) = 1 - \frac{|T(R_d)|}{|\mathrm{R_d}|} \text{ if } |\mathrm{R}_d| > 0 \text{ else } 1$$

$$CON_1(\mathrm{R_d}) = \frac{1}{3} * \left(CON_1(\mathrm{R_d}) + CON_2(\mathrm{R_d}) + CON_3(\mathrm{R_d})\right)$$

$$\mathrm{AFF}(\mathrm{R_d}) = \frac{|A(R_d)|}{|\mathrm{R_d}|} \text{ if } |\mathrm{R_d}| > 0 \text{ else } 1 \quad (3)$$

$$\mathrm{BOU}(\mathrm{R_d}) = \frac{|R_d \setminus R_s|}{|\mathrm{R_d}|} \text{ if } |\mathrm{R_d}| > 0 \text{ else } 1 \quad (4)$$

TABLE III. EXPLANATION OF THE METRICS

Element	Explanation
R_d	Set of requirements, which should be considered
R_s	Not absolutely necessary right now
$A(R_d)$	Set of feasible requirements
$C(R_d)$	Set of requirements with conflicts
$R(R_d)$	Set of duplicated requirements
$T(R_d)$	Set of requirements in which introduced terms are not used consistently

TABLE IV. EXPLANATION OF THE RESULTS

Result	Explanation
1	The quality characteristic is completely fulfilled
< 1	The quality characteristic is not completely fulfilled

B. Identification of Stakeholders

In the elicitation phase, all stakeholders of the project have to be identified. A missing stakeholder can lead to incomplete requirements, which endanger the project success. For this purpose, we apply the approach by Sharp et al. [8]. Based on the four groups a) users, b) developers, c) legislators, and d) decision-makers, for the Participation Service, we could identify all stakeholders as listed in Table V and assign them to the corresponding scrum role.

TABLE V. STAKEHOLDERS OF THE PARTICPATION SERVICE

Group	Stakeholders
Users	Enrolled students and members of the KIT
Developers	Students at C&M and KIT as operator of the Participation Service
Legislators	State of Baden-Wuerttemberg and Federal Republic of Germany
Decision-Makers	C&M leader, C&M members and one expert of systemic consenting

The user represents people, groups, or organizations, which interact with the system or make use of the provided information. The developers are the stakeholders of the requirement engineering process, such as analysts or operators. The legislators represent government authorities that provide guidelines for the development and operation of the Participation Service. The last group stands for the development manager and the user manager, who have the power to make decisions with regard to the characteristics of the system in development.

Depending on the quantity of the stakeholders, a prioritization step is sometimes necessary to assess the importance of the elicited requirements regarding to the influence of the stakeholder. For this reason, Sharp et al. [8] provide an outlook how network theories can be used to determine the influence of a stakeholder. But, such approaches can be time-consuming. A more pragmatic method is the usage of power-interest grid by which the stakeholders are classified in quadrants [22].

In this project, the prioritization of the stakeholders with regard to their influence on the project was not necessary at this point. Due to the fact that the complexity of the project and the amount of involved stakeholders is not as high as in an industrial project.

C. Elicitation of Goals

After the identification of stakeholders, the elicitation of goals can be initiated. For this purpose, the interview and brainstorming technique was chosen and the questions introduced by Bruegge and Dutoit [13] were used for easier discovery of the goals according to the definition by [12], which is shown in Fig. 3. Each goal corresponds exactly to one requirement in order to fulfill the singularity according to [6]. An excerpt of the determined goals is shown in Table VI. Goal G2 will be further refined in the upcoming sections.

In contrast to traditional software methodologies, such as the waterfall approach, in agile development, more goals can be added in the course of the software project.

TABLE VI. EXCERPT OF GOALS OF THE PARTICIPATION SERVICE

ID	Goal	Stakeholder
G1	Logs in at the Participation Service	C&M member
G2	Rate a suggestion	C&M member
G3	Add a new proposal for solution	C&M member

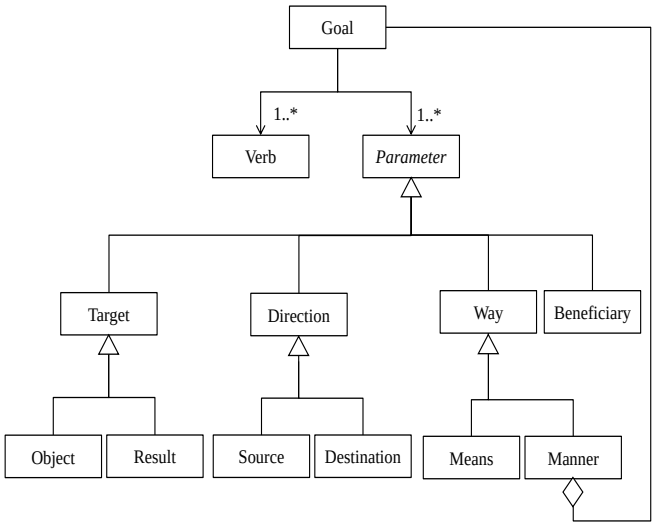


Figure 3. Meta-model of a goal.

By investigating the quality characteristic of the current standard [6], we discovered that the meaning was changed compared to IEEE Std 830-1998 [5]. In [5], requirements were expected to be complete for the entire system. According to the current standard, a set of requirements contains everything to define a system or only a system element. This allows us to use iterations in which system elements are described.

D. Prioritization of Goals

The next step is the prioritization of the goals with regard to their importance for the stakeholders. Due to the abstraction level of the goals and the statement by Wiegers [15], we applied a simple classification approach based on a three-level scale that is shown in Table VII according to IEEE Std 830-1998 [5]. In order to prevent ambiguousness, each stakeholder has agreed on the meaning of each level [15]. After rating of goals, a specific amount of highest ranked goals, which reflects the necessity [6], form the basis for the first iteration. The amount depends on the estimated velocity of the development team and expected effort for the implementation. In this context, the essential goals are those presented in Table VI.

TABLE VII. CLASSIFICATION FOR GOAL PRIORITIZATION

Group	Meaning
Essential	Essential for the next release
Desireable	Not absolutely necessary right now
Optional	Would be nice to have someday

E. Functional Requirements

For each selected goal, a scenario will be authored or reused that describes the required interactions to reach the goal. Based on a scenario, further goals can be derived. The combination of a goal and the corresponding scenario is called requirement chunk as described in [12].

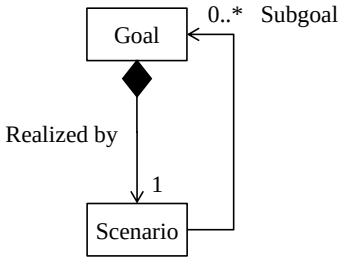


Figure 4. Meta-model of a requirement chunk.

Fig. 4 illustrates this by showing a meta-model that defines the rules and the elements of a requirement chunk. This recursive process with objective of functional decomposition can be aligned with the process defined in the standard [6]. But, this recursive process can be repeated several times, which results in rising costs.

For that reason, we propose three conditions that serve as abort criteria for the process. If all of the following conditions apply, the process can be aborted:

1) no additional benefit in form of new derived goals
2) other scenarios will definitively not reuse atomic actions of the current scenario
3) the size of the scenario exceeds more than 20 atomic actions

According to Glinz [16], the decomposition in user functions and the ease of understanding assure the precondition of correct specification. Furthermore, the decomposition allows us to describe the capability and properties of a given requirement chunk in detail according to the stakeholder's need, which represents the completeness of individual requirements. In the following, authoring and reusing of scenarios will be presented.

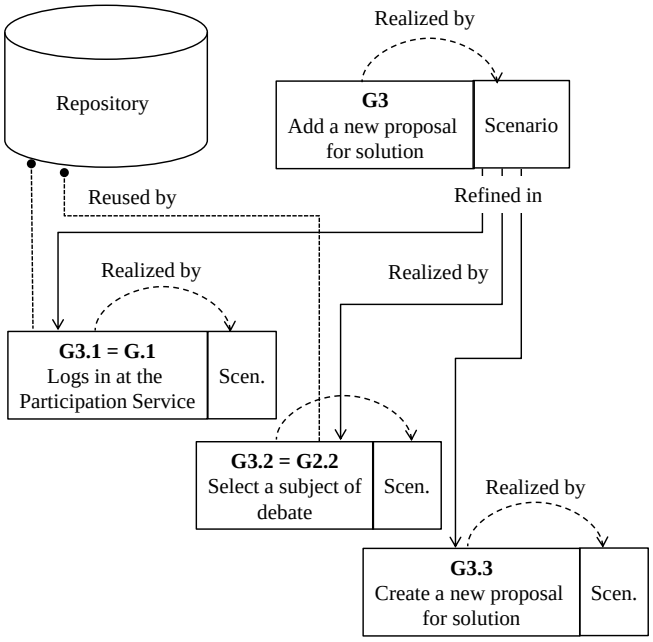


Figure 5. Reusing a requirements chunk from the repository.

E.1. Reusing Scenarios

In the best case, a requirement chunk still exists in the repository, which contains all analyzed goals and their scenarios. Therefore, redundant scenarios will be avoided, which ensures the consistence regarding to a set of requirements. As a result, we can compose different requirement chunks to support higher goals. For example, the goal *G1 "Logs in at the Participation Service"* represents a cross-sectional goal, which will be used by *G2* and *G3*. Fig. 5 shows how the goal *G3* is refined in three different sub goals, while two of them will be reused from the repository.

E.2. Authoring Scenarios

If no requirement chunk for the given goal can be found in the repository, a new scenario has to be authored while considering the quality characteristics by [6].

The unambiguousness cannot be fulfilled properly as we use the natural language with inherent equivocality for the description of the scenario [5]. So a trade-off between ease of understanding and formalism has to be made. For this, we used the provided meta-model of a scenario by Rolland et al. [12] to reduce equivocality, which is shown in Fig. 6 Moreover, we used the introduced structural constructs of Glinz [16] to further reduce the level of equivocality. To detect ambiguousness during description or validation of scenarios, Terzakis [17] offers a detailed checklist. Also, the current standard [6] provides some terms, such as superlatives or vague pronouns, which should be prevented to ensure bound and unambiguousness. For newly introduced terms and units of measure, we have created a separate document, which acts as a glossary.

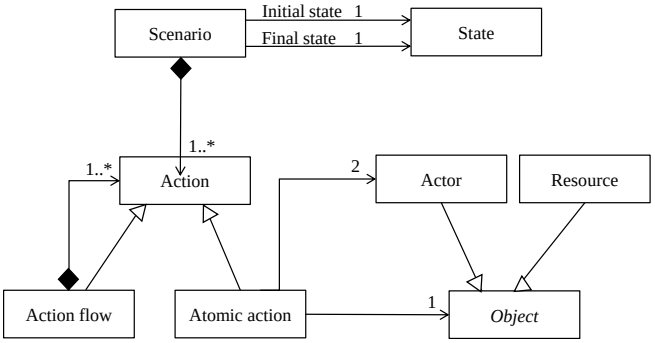


Figure 6. Meta-model of a functional scenario.

According to [6], a scenario should be implementation free. This means that no architectural design decisions take place in this phase. This is the nature of a scenario as it describes what is needed in form of a concrete instance to achieve its intended goals. The nature of a scenario also allows us to derive acceptance criteria to verify the requirements in the form of test cases [16], which fulfills the verifiability [6].

The feasibility is another quality characteristic of the standard [6] with focuses on technical realization of the requirement. At this point, the scenario has to be investigated with regard to system constraints such as the existing environment (cf. Section G).

To ensure the traceability [6], each scenario must have a unique identifier. In the course of modification over time, the scenarios also need a version number representing the current state.

Title:	Rate a proposed suggestion	ID:	G2	Priority:	High
Source:	C&M member	Risk:	Middle	Difficulty:	Nominal
Rationale:	Integral ingredient of systemic finding	Version:	1.0	Type:	Functional

Initial state:	User wants to rate a proposed solution
Final state:	User rated a proposed solution
Dependable goals:	None

No.	Normal action flow	Ref.
1	User logs in at the Participation service	G1
	System verifies the credentials	
2	System redirects him to the secured area (Def. 1.1)	-
	User gets a list of available subjects of debate	
3	User selects a subject from the provided list	-
	System receives the selection and redirects him to the subject of debate	
4	User rates a proposed solution by selecting the refusing points	G5
	System calculates the acceptance of the suggested solution	
No.	Concurrency / Alternative action flow	
2'	*IF* the list of available subjects is empty *THEN* the system displays: There are currently no subjects of debate *TERMINATE*	

Figure 7. Style for representation of scenarios.

Due to the fact of reusing scenarios, each scenario should also be aware of dependable requirement chunks to clarify, which requirement chunks will be affected by modifications of one scenario.

Based on these findings, the representation in [16], and the provided requirement attributes in [6], we created a style for representation of scenarios, which is illustrated in Fig. 7. Similar to the approach by Glinz [16], the representation can also be easily transformed into a state chart.

F. Non-Functional Requirements

After all goals have been analyzed, the resulting requirement chunks represent the functional aspects of the system. Each scenario can now be investigated with regard to non-functional aspects. For this purpose, we use quality attribute scenarios by Ozkaya et al. [18] and link these with the corresponding requirement chunk. The meta model for quality attribute scenario is shown in Fig. 8.

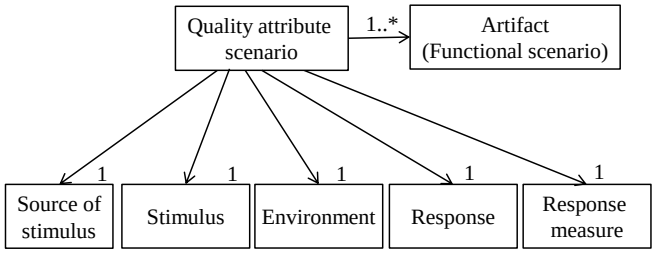


Figure 8. Meta-model of quality attribute scenario.

The stimulus represents the condition for the release of the event, while its source is the entity that triggers it. The response is the activity of the stimulus. The environment, such as normal operation of a service, stands for the constraint under which the stimulus occurred. The functional scenario represents the stimulated artifact. Finally, the response measure represents the measure for evaluating the response of the system.

To align this with the product strategy, the product quality characteristics of ISO/IEC 25010:2011 [11] have to be ranked by their importance for the stakeholders. For example, the security is probably more important than the user experience for a product in the banking sector. This is why we used pairwise comparisons of the quality attributes according to the Analytical Hierarchy Process (AHP) by Saaty [19].

If quality characteristic *A* is more important_than *B*, we assign *A* the value 2 and *B* the value 0. If *A* and *B* are equally important, we assign each of them the value 1.

We took the results of each stakeholder and calculated the average, which is shown in Fig. 9. As Fig. 9 shows, security, functionality, and usability are more important than the others. Based on this result, we could focus on the most important quality attributes. Nevertheless, we still have to keep the quality attributes with minor importance for the product strategy in mind. We can thus reduce the effort for eliciting the non-functional requirements since resources, such as time, often limit a project.

	Functional suitability	Performance efficiency	Usability	Compatibility	Reliability	Security	Maintainability	Portability	*Sum*	*Weight*
Functional suitability		11	6	10	7	6	9	11	60	0.18
Performance efficiency	1		0	4	2	0	2	7	16	0.04
Usability	6	12		10	8	3	9	11	59	0.18
Compatibility	2	8	2		4	2	3	8	29	0.09
Reliability	5	10	4	8		3	5	7	42	0.13
Security	6	12	9	10	9		9	12	67	0.20
Maintainability	3	10	3	9	7	3		10	45	0.13
Portability	1	5	1	4	5	0	2		18	0.05
Sum									336	1.0

Figure 9. Results of the Analytical Hierarchical Process (AHP).

Similar to the description of the functional scenarios (c.f. Section E), we have to respect the same conditions. This is why we do not describe this in detail at this point.

For the prioritization of non-functional requirements, we used the ranked result of the AHP. But, it is also possible to add another prioritization step, such as the ones mentioned in [15] or [18]. Fig. 10 shows one non-functional requirement of goal *G2*.

Type:	Usability	ID:	N2	Priority:	0.18
Source:	C&M member, students	Risk:	Low	Difficulty:	Easy
Rationale:	Better user experience	Version:	1.0	Reference:	G2

Quality attribute scenario	*Source of stimulus:*	User
	Stimulus:	clicks on the button
	Environment:	during normal operation,
	Response:	the system gives a feedback
	Response measure:	within a period of 200ms

Figure 10. Style for representation of quality attribute scenarios.

G. Constraints

According to Glinz [10], the constraints restrict the solution space for the functional and non-functional requirements. For example, a constraint can be company-based human interface guidelines, legal issues, or existing environments [10]. With regard to the Participation Service, we only had to investigate the constraints emerging from the existing environment. As described in the introduction, the Participation Service should be a part of the existing service-oriented SmartCampus System based on REST.

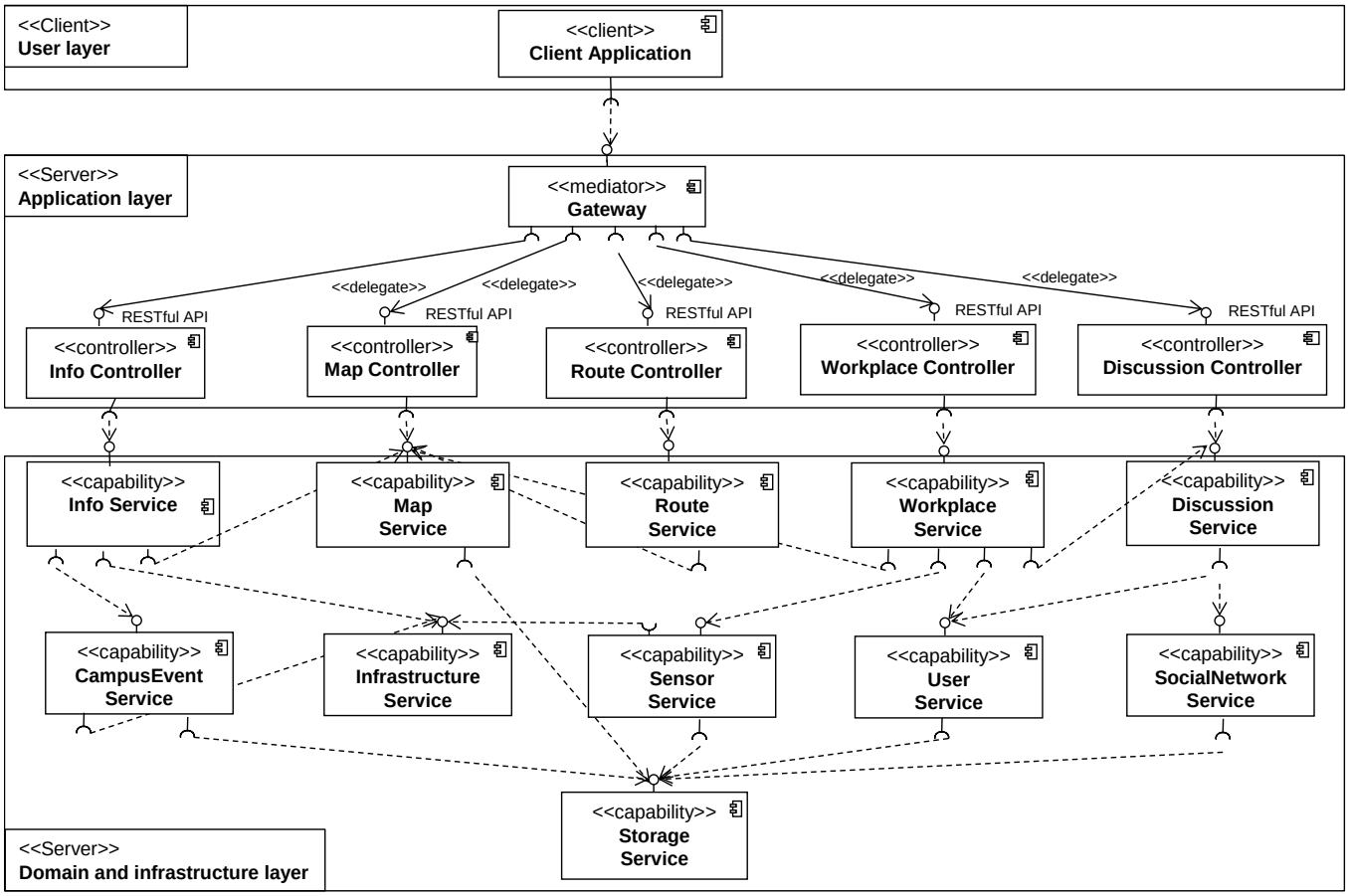


Figure 11. Component diagram of the SmartCampus system at the KIT.

Fig 11 shows the layered architecture according to Evans [23] with components of the current SmartCampus System, which consists of four layers: 1) user layer, 2) application layer, 3) domain, and 4) infrastructure layer. The latter ones are combined in the illustration for a better overview.

REST is a hybrid architectural style for distributed hypermedia systems according to Fielding [2], which he defines as follows: "REST is a hybrid style derived from several of the network-based architectural styles ... and combined with additional constraints that define a uniform connector interface." [1, p. 76]. This definition implies the consideration of several constraints that can be segmented in architectural (1 - 5) and interface constraints (6) [1][21]:

1) Client-Server indicates a client and server component. The client component sends a request to the server that should be performed. Based on the request, the server component either rejects or performs the request.
2) Statelessness avoids the need of maintaining information about a previous request on server side. This leads to an improvement of server scalability.
3) Caching avoids a replication of already transmitted information over the network.
4) Layered architecture facilitates the usage of mediator components for adding features such as load-balancing.
5) Code on demand is an optional constraint, which extends the client functionality at runtime trough downloading an executable artifact.
6) Uniform interface is an "umbrella term for the four interface constraints" [21, p. 356]: the identification of resources, the manipulation of resources through representation, the self-descriptive messages und the hypermedia constraint.

These constraints were written down in a separate constraints document similarly to the glossary so that we are able to reference this over the whole iteration cycle with regard to the feasibility [6].

H. Verification

After the elicitation of the requirements in a quality-oriented way, we have investigated the requirements according to the formalized characteristics for a set of requirements in Section IV. These results give us a hint to what extent the elicited requirements fulfill the quality characteristics of ISO/IEC/IEEE 29148:2011 [6].

TABLE VIII. RESULTS OF THE VERIFIED SET OF REQUIREMENTS IN EACH PERFORMED ITERATION

Metric	Iteration #1	Iteration #2	Iteration #3	Iteration #4
COM(R_d)	0,97	0,92	0,93	0,99
CON(R_d)	1,0	1,0	1,0	1,0
AFF(R_d)	1,0	1,0	1,0	1,0
BOU(R_d)	1,0	1,0	1,0	1,0

Based on the results in Table VIII, we could prove our assumption that the full set of quality characteristics can be fulfilled by ensuring the individual ones. The only exception is the completeness, which was already mentioned in Section IV. Because of this, we recommend the investigation of the completeness before designing and implementing the specified system or system element to ascertain the quality of the requirements.

V. EVALUATION

Our results by applying this technique showed us that we improved the quality of our requirements by using this technique, which considers the quality characteristic of ISO/IEC/IEEE 29148:2011 [6]. For example, we have detected some inconsistencies during the authoring of the scenarios and reduced the communication effort emerged from misunderstandings.

Compared to the previous recommendation [5], it is easier to meet the desired qualities of ISO/IEC/IEEE 29148:2011 [6]. The reason for this is that the new standard does not give tough specifications for the satisfaction of the quality characteristics.

Due to the fact that we are using the natural language for describing requirements, we can only merely reduce the ambiguousness and not prevent completely. However, this does not imply bad requirements but rather potential for improvements. Furthermore, sometimes it is adequate to achieve 90 percent of the quality criteria, because the cost to reach 100 percent is too high.

Furthermore, we propose the adjustment of the completeness so that partial specifications in form of iterations are allowed. The precondition of the completeness will be analyzed with regard to the goals of the current iteration.

VI. CONCLUSION AND OUTLOOK

In this article, we introduced a methodology for requirements engineering of RESTful web service for systemic consenting. The methodology ensures that the requirements fulfill quality characteristics defined by the international standard ISO/IEC/IEEE 29148. For that purpose, we analyzed existing methodologies and combined those parts that consider a certain quality characteristic to a new methodology. Thus, the methodology presented in this article is a combination of existing work.

As stakeholders and participants often have different opinions, it is necessary to find consensuses. For that purpose, the Participation Service implements functionality that is based on the concept of systemic consenting. By applying the requirements engineering methodology presented in this article, the quality of the requirements for the Participation Service could be improved. For example, we detected some inconsistencies during the authoring of the scenarios and reduced the communication effort and the costs emerged from misunderstandings.

Compared to the previous IEEE Std 830-1998 [5], it is easier to meet the desired qualities of ISO/IEC/IEEE 29148 [6]. The reason for this is that the new standard does not give tough specifications for the satisfaction of the quality characteristics. Due to the fact that in a scenario-based approach we are using the natural language for describing requirements, we can only merely reduce the ambiguousness and not prevent it completely. However, this does not imply bad requirements but rather potential for improvements.

Our approach is currently focused on the Participation Service and its specifics. We assume that the methodology is also applicable for further services or even software systems in general. However, this is not proven yet. With this approach, we expect to support requirements engineers and business analysts when they have to describe the requirements for a RESTful web service. In our scenario, the presented methodology helped with gathering and describing functional and non-functional requirements in a systematic way so that they are of high quality. As the quality characteristics considered in this article are part of an international standard, they can be seen as valid and of importance. Furthermore, requirements engineers and business analysts can apply this methodology to analyze and improve already described requirements regarding their quality. As the requirements constitute the basis for the rest of the development process, it is of high importance that a certain level of quality is reached. For that reason, when generalizing this approach, it will contribute to the development of high-quality software solutions.

For the future, before generalizing the approach, we plan to focus on further parts of the development of high-quality RESTful web services. With this article, we considered the initial phase of the development process, the gathering and description of requirements. In the next step, we will focus on the design of RESTful web services that fulfill the previously gathered requirements. Also in this case, the quality of the result will be considered. For that purpose, we will analyze existing best practices for the design of RESTful web services. We will combine these best practices with quality characteristics of ISO 25010:2011 as a standard for the quality for software products. Especially in environments with limited resources, such as time and money, not all best practices can be considered. By associating best practices with quality characteristics, it will be possible to prioritize best practices for the design of RESTful web services and to select the for a certain project most valuable ones. Finally, we aim to enable an automatic measurement of the best practices to rapidly get an impression of the degree of fulfillment.

For that purpose, we will enhance our existing work in the context of quality assurance of service-oriented architectures [20]. We are also already working on an open source tool, the QA82 Analyzer, to automate the measurement of best practices [24]. After focusing on the requirements engineering, the future work will help us to also design and develop the Participation Service and future web services in a quality-oriented manner.

REFERENCES

[1] M. Gebhart, P. Giessler, P. Burkhardt, and S. Abeck, "Quality-oriented requirements engineering for agile development of restful participation service," Ninth International Conference on Software Engineering Advances (ICSEA 2014), Nice, France, October 2014, pp. 69-74.

[2] R. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[3] Standish group, "Chaos report," http://www.projectsmart.co.uk/docs/chaos-report.pdf, 1995, Accessed 2014-05-21.

[4] A. F. Hooks and K. A. Farry, "Customer centered products: creating successful products through smart requirements management," American Management Association, 2000, ISBN 978-0814405680.

[5] IEEE, IEEE Std 830-1998 "Recommended practice for software requirements specifications," 1998.

[6] ISO/IEC/IEEE, ISO/IEC/IEEE 29148:2011 "Systems and software engineering – life cycle processes – requirements engineering," 2011.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," The Future of Software Engineering, Special Volume published in conjunction with ICSE, 2000, pp. 35-46.

[8] H. Sharp, A. Finkelstein, and G. Galal, "Stakeholder identification in the requirements engineering process," Database and Expert Systems Applications, 1999, pp. 387-391.

[9] F. Ackermann and C. Eden, "Strategic management of stakeholders: theory and practice," Long Range Planning, Volume 44, No. 3, June 2011, pp. 179-196.

[10] M. Glinz, "On non-functional requirements," 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, pp. 21-26.

[11] ISO, ISO/IEC 25010:2011 "Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - system and software quality models," 2011.

[12] B. C. Rolland, C. Souveyet, and C. B. Achour, "Guiding goal modeling using scenarios," IEEE Transactions on Software Engineering, Volume 24, No. 12, 1998, pp. 1055-1071.

[13] B. Bruegge and A. H. Dutoit, "Object-oriented software engineering: using uml, patterns and java," Pearson Education, 2009, pp. 166-168.

[14] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," IEEE Software, Volume 14, No. 5, 1997, pp. 67-74.

[15] K. Wiegers, "First things first: prioritizing requirements," Software Development, No. 9, Volume 7, Miller Freeman, Inc, September 1999, pp. 48-53.

[16] M. Glinz, "Improving the quality of requirements with scenarios," Proceedings of the Second World Congress on Software Quality, Yokohama, 2000, pp. 55-60.

[17] J. Terzakis, "Tutorial writing higher quality software requirements," ICCGI, http://www.iaria.org/conferences2010/filesICCGI10/ICCGI_Software_Requirements_Tutorial.pdf, 2010, Accessed 2014-07-16.

[18] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan, "Making practical use of quality attribute information," IEEE Software, April 2008, pp. 25-33.

[19] T. L. Saaty, "How to make a decision: the analytic hierarchy process," Informs, Volume 24, No. 6, 1994, pp. 19-43.

[20] M. Gebhart, "Measuring design quality of service-oriented architectures based on web services," Eighth International Conference on Software Engineering Advances (ICSEA 2013), Venice, Italy, October 2013, pp. 504-509.

[21] L. Richardson, M. Amundsen, S. Ruby "RESTful Web APIs," O'Reilly, 2013.

[22] F. Ackermann, C. Eden "Strategic Management of Stakeholders: Theory and Practice," Long Range Planning, Volume 44, No. 3, 2011, pp. 179-196.

[23] E. Evans, "Domain-Driven Design: Tacking Complexity In the Heart of Software," Addison-Wesley Longman Publishing Co., Inc., 2003.

[24] QA82, QA82 Analyzer, http://www.qa82.org, Accessed 2015-02-12.

From Software Engineering Process Models to Operationally Relevant Context-aware Workflows: A Model-Driven Method

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Abstract - **Software engineering (SE)-specific process models and their notation, such as the Software & Systems Process Engineering Metamodel, are typically not specified or available in an executable form that can provide automated guidance in human-centric software engineering workflows. These SE process models generally remain abstract in order to be broadly applicable, and when any are concretized, they often exist only in the form of documentation. Thus, they are not actually relevant operationally, affecting process utilization and governance. On the other hand, common business process modeling notation such as BPMN is generalized and not conducive for providing the context-aware support needed for executable SE workflows. Thus, a practical method is needed that supports comprehensive SE process documentation, yet also provides an SE workflow modeling capability that can transform documented SE workflows into an enactable form executable in today's workflow management systems. The method presented in this paper can utilize an available comprehensive SE process documentation meta-model and automatically extract incorporated SE concepts and workflow concepts to a workflow model, specifically the Software Engineering Workflow Language (SEWL). From this the following are supported: 1) graphical-based workflow modeling, 2) model-based transformation of workflow concepts to diverse workflow management systems (WfMS), and 3) the semantic transformation of SE concepts to contextually-aware process-centered software engineering environments. The results show the viability and practicality of such a method to document, extract, graphically model, transform, and enact SE workflows in support of contextual guidance capabilities for software engineers.**

Keywords - process-centered software engineering environments; software engineering environments; software engineering process modeling; software engineering process model transformation; SPEM; UMA; Unified Method Architecture; model-driven software development.

I. INTRODUCTION

This article extends our previous work in [1]. In order to be generally applicable to various software development projects, most software engineering (SE) process models remain abstract and require tailoring to the specific project, team, and tool environment. Examples of SE process models include the V-Model XT [2] (specified for all public-sector IT development in Germany) and the Open Unified Process (OpenUP) [3]. Typical SE process models are documented to a great extent in natural languages, and are thus not easily executable in an automated form. The technical implementation of an executable process, whose sequence can be modeled with and automatically enacted in a workflow management system (WfMS), is called a workflow. SE workflows, many of which are human-centric, can cover some sequence of activities and steps related to requirements, design, testing, etc., for instance Activity Flows in VM-XT [4] or workflows in OpenUP [5].

Because they integrate SE concepts, SE process meta-models can be useful in the modeling and comprehensive documentation of such SE processes. For instance, the Eclipse Process Framework (EPF) [6] is an open source project for software process engineering that provides a framework and supporting tools, one being the EPF Composer (EPFC) [7] for method and process authoring and publishing. It utilizes a process meta-model, the Unified Method Architecture (UMA), a large extent of which was adopted into the Software & Systems Process Engineering Metamodel (SPEM) 2.0 [8].

Additionally, process-centered software engineering environments (PCSEEs) have attempted to investigate and address automated guidance and assistance mechanisms for SE processes [9]. Yet they remain intrusive, rigid, and inflexible [10], and fail to adequately support the human, creative, and dynamic aspects of software development. While more generalized automated process assistance and guidance for humans has been available in the form of process-aware information systems (PAIS) [11], this area has lacked satisfactory standards and SE support and often lacks the integration of the project and human context. Thus, such systems and capabilities have not been readily leveraged by software engineers.

A. Our Previous Work

To address these challenges for such human-centric SE processes, we created a PCSEE that we call the Context-aware Software Engineering Environment Event-driven Framework (CoSEEEK) [12]. Beyond SE tool sensors and other contextual knowledge, it utilizes workflows to understand the process context. That includes knowing which activities a software engineer performed, which activity is likely currently being worked on, which activity is next, and associating these with SE-specific concepts such as

projects, teams, persons, roles, tools, and artifacts via an ontology and reasoner. While various facets were investigated, including collaboration [13], quality integration [14], and others, we still faced the problem of providing an easy way for software engineers to access, model, and transform SE workflows and integrate SE concepts without vendor lock-in to a specific WfMS. Considering possible SE workflow modeling notation, the SPEM is aimed primarily at defining a domain-specific notation for the documentation of SE processes, and does not completely address issues related to executable SE processes so that automatic support and guidance for software engineers in operational activities can occur. On the other hand, a general workflow language notation such as the Business Process Model and Notation (BPMN) 2.0 [15], while executable, lacks the inclusion and semantic meaning of various SE domain-specific concepts and thus becomes cumbersome.

Thus, to address the executable SE workflow language gap, our team created the text-based language SEWL [16] and previously targeted the adaptive WfMS AristaFlow [17] and YAWL [18] to evaluate its portability. Our previous work in [1] contributed various extensions to the original workflow concepts, including: a new graphical representation for SE-specific workflows blending BPMN and SPEM notation; a graphical editor for SE workflows; details on the model-driven generation of tailored artifacts that target the ontology and heterogeneous WfMS support, specifically the common of-the-shelf (COTS) WfMS jBPM [19] and Activiti [20]; and the workflow ontology generator, which addresses the aspect of contextual-awareness support for workflows in conjunction with CoSEEEK.

B. Contribution

This article extends our work in in [1] by expanding the scope of the original solution approach. It contributes an automated model-driven method for SE process modelers that incorporates a standard SE process meta-model, namely the UMA, thus supporting comprehensive SE process documentation capabilities while generating concrete enactable workflows that can be used in automated SE guidance support. Based on the information gleaned from the SE model, workflow concepts are transformed into an intermediate workflows language SEWL, from which further workflow transformations to specific WfMS can occur. An evaluation utilizes the EPF Composer with both existing and new SE process models and with Activiti and jBPM WfMS, the results showing the viability and practicality of the method for documenting an SE model with existing tooling, extracting SE concepts, graphically modeling SE workflows, transforming SE workflows to specific WfMS formats, and enacting SE workflows in support of contextual guidance capabilities for software engineers.

The summary of the paper is as follows: the following section discusses related work, and Section III describes the solution method. Section IV then describes our realization of the method. Section V presents an evaluation, followed by a discussion and then the conclusion.

II. Related Work

With regard to related work, SPEM 2.0 [21] was approved without supporting full process enactment. It proposes two possible approaches for enactment: One proposes a mapping to project planning tools. However, this does not support automated adaptation to changing project contexts during project execution. The other proposal is to use the Process Behavior package to relate SPEM process elements to external behavior models using proxy classes. Both approaches lack full workflow modeling and executability at the level of BPMN.

Other work related to enactment of SPEM includes eXecutable SPEM (xSPEM) [21]. Process execution is addressed via transformation to the Business Process Execution Language (BPEL), while process validation is addressed via transformation to a Petri net in combination with a model checker. [22] maps SPEM to the Unified Modeling Language Extended Workflow Metamodel (UML-EWM) in order to create a concretely executable workflow. [23] and [24] investigate transforming SPEM to BPMN, while [25] maps SPEM to the XML Process Definition Language (XPDL). xSPIDER ML [26] is an extension profile of SPEM 2.0 to enable process enactment.

The novelty of our solution method is that, in contrast to the above approaches, it targets a simple graphical as well as textual SE process language and notation for modeling, blending the strengths of BPMN and SPEM; it concretely generates executable workflows on different WfMS targets; and it generates an Web Ontology Language (OWL)-compliant ontology of SE concepts for context-aware PCSEE tooling support. This addresses prior hindrances and challenges for modeling and contextually integrating SE workflows in SEE. Furthermore, the model-driven integration of SE process meta-models provides a way to support comprehensive SE process documentation while providing an automated method to extract enactable SE workflows and contextual concepts.

III. Solution

This section describes our model-driven solution method (refer to Figure 1). The description below will refer to the four phases in the method shown at the top of Figure 1, namely *model*, *transform*, *deploy*, and *operate*.

The basis of the solution concept is an SE workflow model, such as SEWL workflows which we had previously developed. While the method supports the use of any SE workflow format, SEWL was used as an intermediate workflow model in our realization of this method. A SEWL workflow is modeled, either with the graphical SEWL editor or a textual editor, and provided as input for our Generator. To transform the input, our Generator utilizes various adapters we created that generate appropriate workflow templates tailored for a specific WfMS, while concurrently providing OWL-DL [27] output of the semantic concept instances. These templates are then deployed. During operations, a Process Manager Service we created abstracts, via an interface, the WfMS-specific integration and interaction details for our CoSEEEK (thus CoSEEEK does

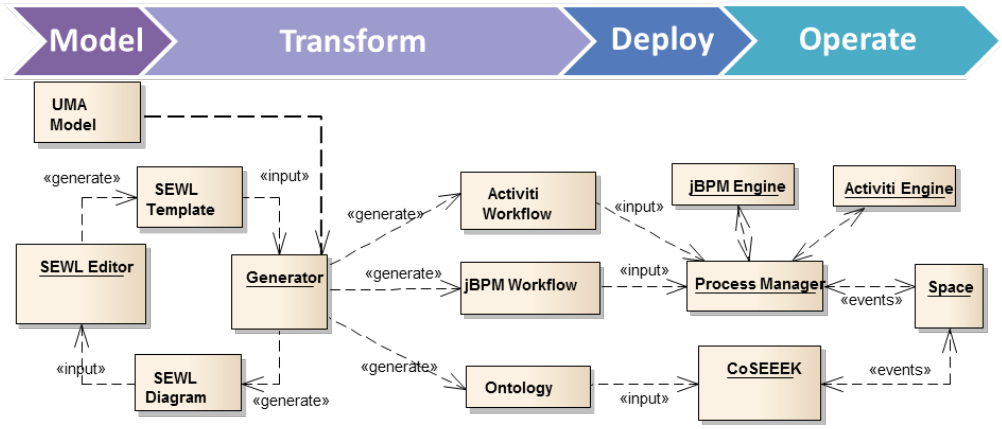


Figure 1. Solution method for SE process model transformation to executable and contextually-aware workflows.

not need to be a PAIS but only extend one) and the ontology is referenced internally during operations by CoSEEEK. Ontologies and semantic technology are advantageous in providing a taxonomy for modeled entities and their relations, a vocabulary, and supporting logical statements about entities [28]. Automated consistency checking and interoperability between different applications and agents also support SE environment concept reuse.

A. The Model Phase

In the *model* phase (see Figure 1), an SE process is modeled and documented based on an SE meta-model, such as a *UMA model* created using the EPF Composer. This model serves as an input to our Generator during the *transform* phase, which automatically generates a SE workflow model that maps all the correlating SE concepts to an intermediate SE workflow format such as SEWL. Alternatively, one could model or adapt the workflows directly in SEWL. In the process modeling phase, a graphical *SEWL Editor* assists the process modeler in creating the textual SEWL workflows, which maintain the essence of workflow concepts. Supplemental graphical diagram information (position, font, color, etc.) is retained in separately maintained diagram files, which are kept in sync with the SEWL workflows. Direct editing of the XML-based SEWL format is also possible; however, the Generator will remove all non-applicable elements from the graphical *SEWL diagrams* since the *SEWL template* XML file is considered the primary model source. Further details are provided in the next section.

B. The Transform Phase

As shown in Figure 1, in the *transform* phase SE workflow model inputs in a format such as SEWL are transformed by a Generator with a plug-in transformation adapter architecture to the executable workflow template format of a given WfMS target. An OWL adapter in the Generator also semantically transforms SE concepts in the workflow to produce an OWL-DL compliant ontology that it utilized for process contextual awareness by CoSEEEK.

To exemplify what the *transform* phase of our method does, Table I shows the mapping of common SE workflow concepts. Here WU stands for Work Unit and WUC for Work Unit Container. The primary difference between jBPM and Activiti concept mapping is that in Activiti loops are typically expressed via inclusive gateways and in jBPM via exclusive gateways. E.g., any concurrent tasks in an SE workflow would be modeled with the BPMN parallelGateway, which activates all branches simultaneously and, when merging, waits for all branches to complete. Most WfMS support such basic features.

TABLE I. MAPPING OF SE AND WORKFLOW CONCEPTS

SEWL	Activiti	jBPM	Ontology
Phase	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Activity	Service Task	Service Task	WUC + WU
Iteration	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Task	Service Task	Service Task	WU
Sequence	-	-	-
Parallel	parallelGateway	parallelGateway	-
Loop	inclusiveGateway	exclusiveGateway	-
XOR	exclusiveGateway	exclusiveGateway	-
Roles	-	-	Role Template
Artifacts	-	-	Artifact Template
Variables	-	-	Workflow Variables Template

To address and abstract the integration, communication, and coordination details of the specific WfMS, each Activity or Task is represented as a Service Task and, during generation, wrapped with code that supports the tracking or triggering of the start and finish of an activity or task via event sources and event listeners. This is done since a Process Manager Service abstracts the integration specifics of a WfMS for CoSEEEK, and a Space (a tuple space [29]) we developed is used to handle loosely-coupled communication during operation with CoSEEEK. Details on this are provided later in this and the next section.

C. The Deploy Phase

In the process deployment phase shown in Figure 1, workflows in a WfMS-specific format are deployed into their respective WfMS engine (e.g., jBPM or Activiti) and the workflow ontology deployed into CoSEEEK. Typically this implies transferring the workflow files to the expected locations for a given configuration.

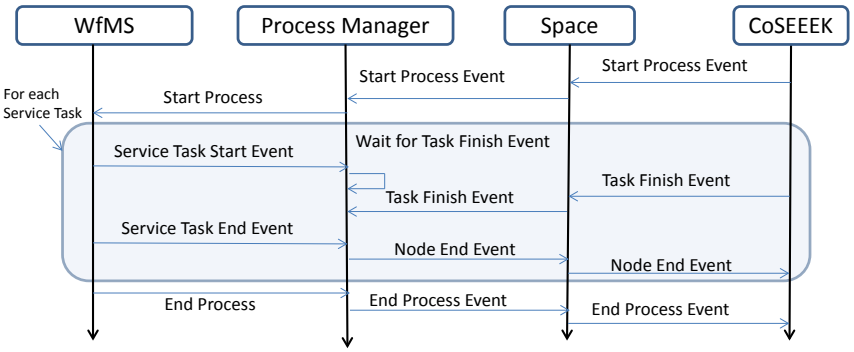


Figure 2. Primary runtime component interaction.

D. The Operate Phase

In the *operate* phase, a specific WfMS instantiates and executes workflow instances as prescribed by a Process Manager Service, which integrates a WfMS and abstracts its details and peculiarities. In our method, to support heterogeneous WfMS a Process Manager acts as an intermediary to support indirect and loosely-coupled event-based interaction between a Service Task and the context-aware PCSEE. In our method implementation, CoSEEEK uses our Space for this. Note that the *transform* phase needs to incorporate the expected operational semantics in order to generate the appropriate workflows for the operational phase.

Figure 2 provides an example of the operational interactions in our implementation of the method. The Process Manager has registered as a listener for certain events. CoSEEEK writes a Start Process Event into the Space. The Space notifies the Process Manager of this event, which in turn instantiates and starts a given process in the WfMS. For each Service Task in the workflow, a Service Task Start Event is sent to the Process Manager and the task waits until further notice. When CoSEEEK becomes aware of a context state change via tool sensors (e.g., a commit of source code was done, a test was started, or the software engineer manually chose a new activity) that affects this workflow and indicates that the current task is completed, CoSEEEK writes a Task Finish Event to the Space. The Space notifies the registered Process Manager of this event, and it in turn notifies the WfMS. The Service Task sends an End Event when it completes, and the Process Manager write a Node End Event in the Space. The Space notifies CoSEEEK of the event, which updates its state. When the final Service Task completes, the workflow completes, and the Process Manager writes an End Process Event to the Space, which notifies CoSEEEK, which in turn updates its state. As an aside, because all event history is kept in the Space, any CoSEEEK components or a Process Manager coming online after some absence (e.g., restart) can determine the context or catch up on any missed events.

IV. REALIZATION

This section provides details on the implementation of our method, the current contribution being primarily the integration of UMA support. To support loose-coupling with CoSEEEK, a service-oriented event-driven architecture was used in conjunction with a tuple space [29] composed on top of a native XML database eXist [30], and provides a Web Service for remote access. A Process Manager Service manages and abstracts the peculiarities of a WfMS, interacting indirectly with CoSEEEK via events in the Space.

The Eclipse Graphical Modeling Framework (GMF) [31] and Eclipse Modeling Framework (EMF) [32], which includes ecore, were utilized by the SEWL editor. Figure 3 shows a simplified snippet of the ecore-based metamodel.

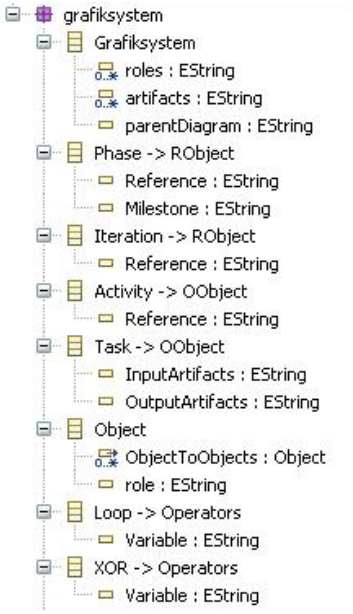


Figure 3. Simplified portion of the metamodel used with ecore.

Transformation Adapters. The Generator and associated pluggable transformation adapters (SEWL, UMA, OWL, jBPM, Activiti, AristaFlow, YAWL) were realized primarily in Scala. Unique IDs were generated for every element transformed and its target transformed element. This permits a clear mapping association, which is also useful for logging. The ontology adapter uses the Jena framework for programmatic ontology access [33] to generate the ontology instances for phases, activities, roles, artifacts, etc.

Figure 4 shows an example workflow snippet generated for a jBPM Service Task, while Figure 5 shows one for Activiti. For details on XML grammar of inputs or outputs, refer to the respective WfMS or UMA documentation.

```
<task id='2'name='RequestChange'tns:taskName='SEWL Task'>
 <extensionElements>
 <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>StartEventListener listener = new
StartEventListener();
     kcontext=listener.writeNodeStart(kcontext);</script>
 </tns:onEntry-script>
<tns:onExit-script
scriptFormat='http://www.java.com/java'>
  <script>EndEventListener listener = new
EndEventListener();
           listener.writeNodeEnd(kcontext);</script>
  </tns:onExit-script>
 </extensionElements>
</task>
```

Figure 4. Listing snippet of generated jBPM Service Task.

```
<serviceTask id='RequestChange' name='Request Change'
activity:class='Service'>
 <extensionElements>
  <activity:executionListener event='start'
class='StartEventListener'/>
  <activity:executionListener event='end'
class='EndEventListener'/>
 </extensionElements>
</serviceTask>
```

Figure 5. Listing snippet of generated Activiti Service Task.

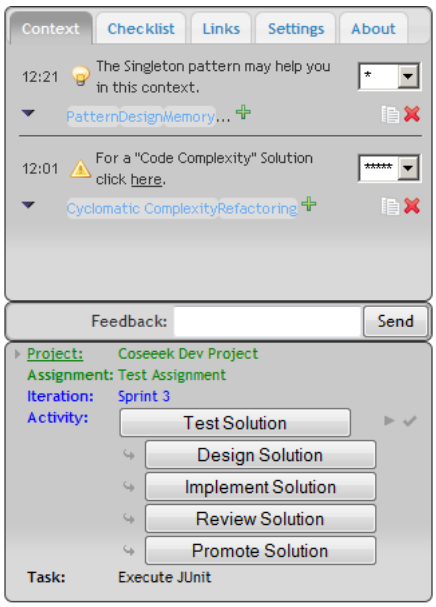


Figure 6. CoSEEEK guidance GUI.

Generated OWL output was loaded into the Protégé ontology editor [34] and is shown for a work unit activity in Figure 7. Because the entire XML is very verbose, it is not shown. Figure 8 shows a small portion of the CoSEEEK software engineering environment ontology in graphical form to give an impression of how software engineering environment concepts, properties, and relations, such as work units and activities are tied into the larger project and environment, which is then utilized to provide contextual awareness.

These workflows and the associated ontology concepts serve as input to CoSEEEK, which then can provide contextual guidance for a software engineer during SE process execution, as can be seen in the screenshot of the HTML- and JavaScript-based CoSEEEK GUI (Graphical User Interface) shown in Figure 6. Context notifications are shown in the upper region, and process context guidance is in the bottom region, showing the current workflow activity (here 'Test Solution') and the next possible follow-on activity choices for guidance and/or manual selection by the user [35]. Details on CoSEEEK's holistic approach to support contextual guidance [12] and collaboration [13] during the SE process are published in other papers and beyond this paper's scope. For example, [14][36][37][38] provide details on the automated integration of software quality measures into executing SE workflows.

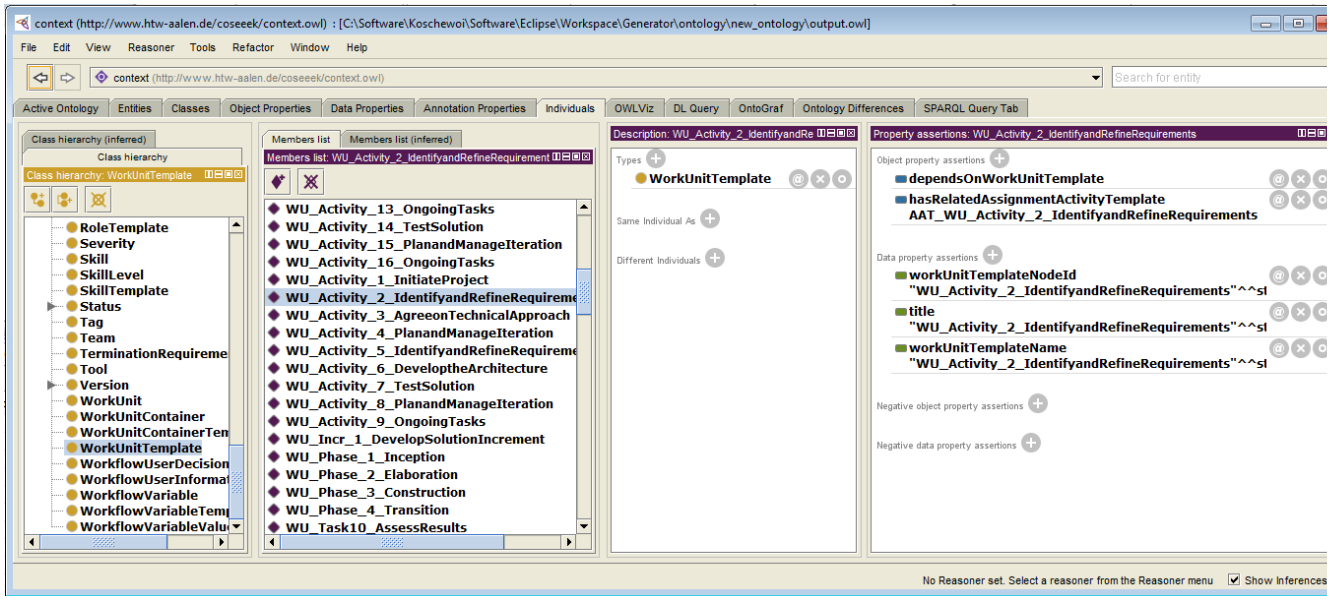


Figure 7. Generated OWL ontology for CoSEEEK shown in Protégé.

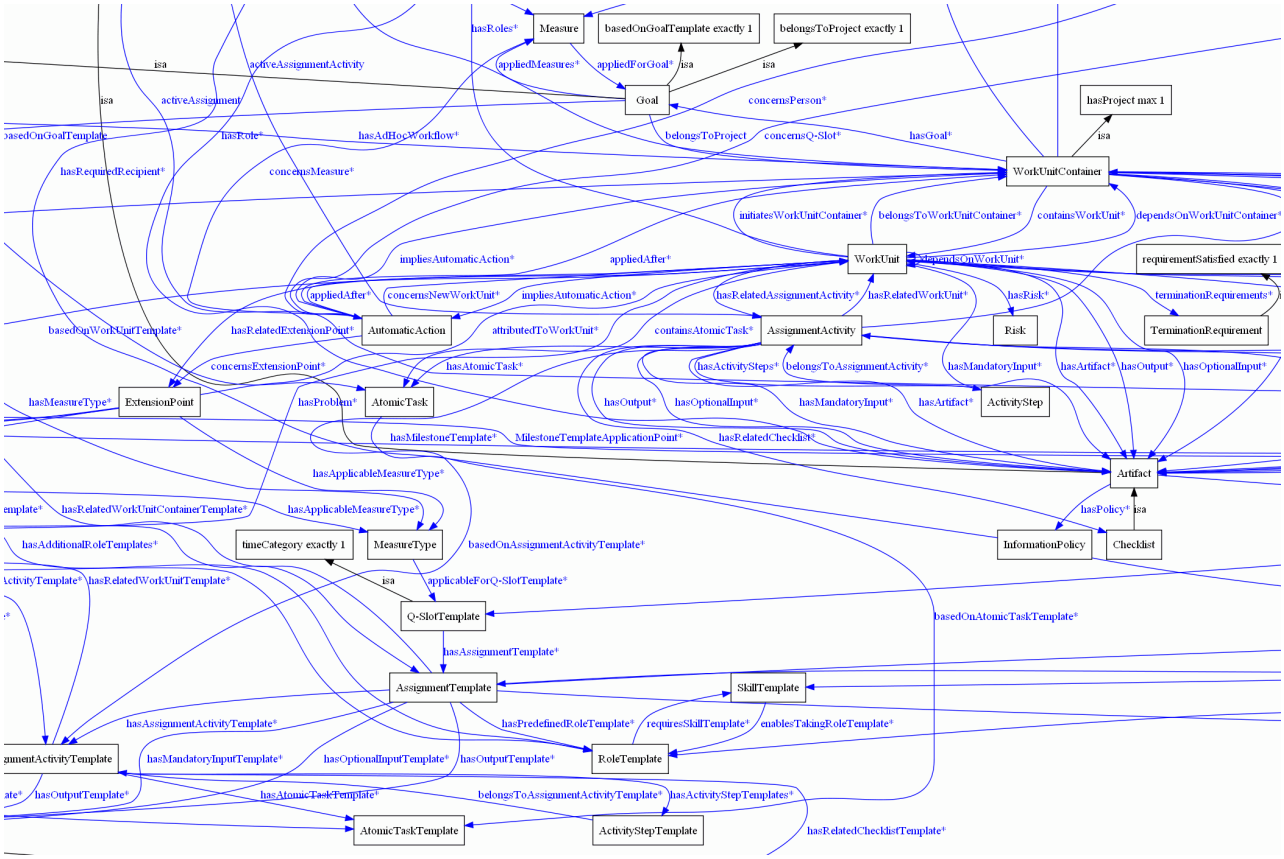


Figure 8. Screenshot of a portion of the CoSEEEK's software engineering environment ontology.

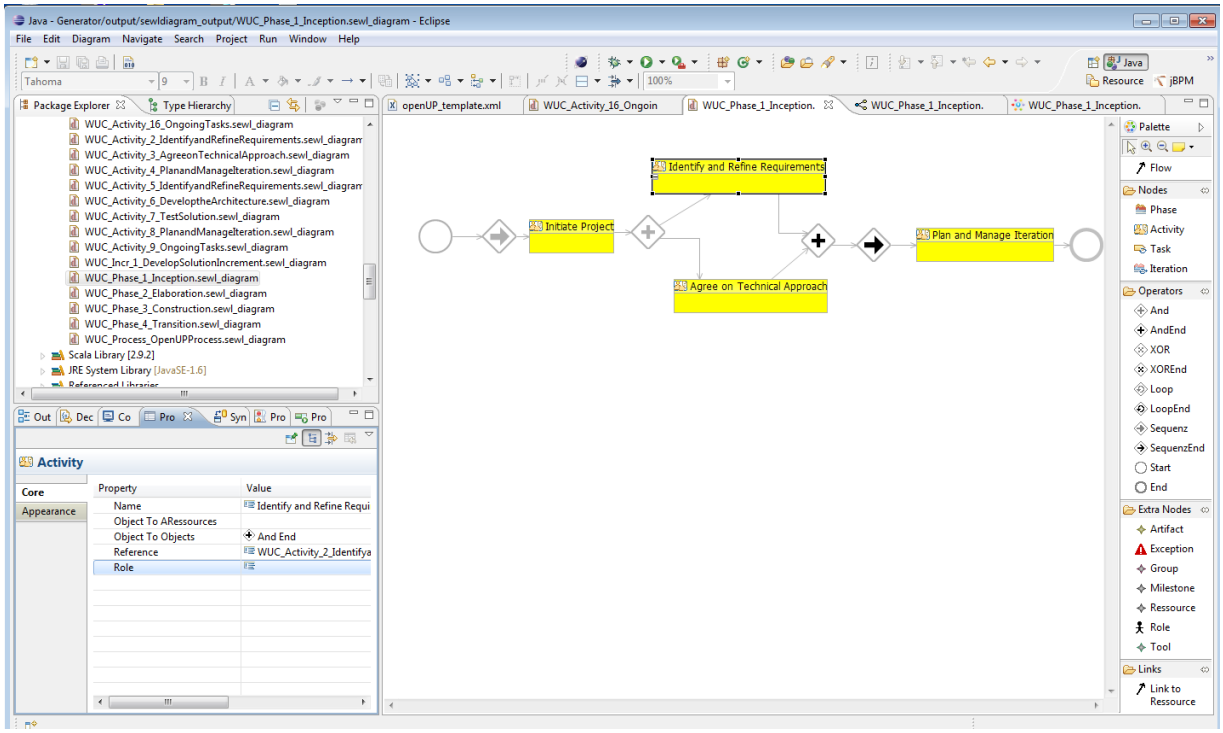


Figure 9. SEWL Editor showing the OpenUP Inception Phase in the SEWL graphical format.

SEWL Editor. The SEWL textual language described in [16] supports the modeling of SE workflow concepts that a SE process may have. Multi-lingual support for referencing the same SE concept instance in various natural languages (e.g., German and English) was also implemented previously, supporting global software development (GSD) processes and their documentation in multiple languages.

The graphical notation used in the editor is extensible and can be adapted or "skinned" with icons to suit the preferences of the user, which can minimize notation confrontations between different user "tribes", e.g., BPMN purists or SPEM purists. In order to get the "best of both worlds", the SEWL Editor currently applied a mix of graphical notation as follows:

- SPEM icons for all SE concepts (e.g., phase, activity, iteration, task, role, artifact),
- BPMN icons for process notation, e.g., events, gateways, and connections.

As an example, an OpenUP Inception phase workflow modeled in the SEWL Editor is shown in its graphical (Figure 9) and textual (Figure 10) notation. One can see that various SE concepts such as roles, phases, artifacts, activities, inputs, and outputs can be modeled and sequenced.

```
<process base="default_process.xml" xmlns=...>
  <resources>
    <roles>
      <role id="1" name="Analyst" />
      <role id="2" name="Project Manager" />
...
  <elements>
    <element name="phase" base="container">
      <structure>
        <attribute name="repeatable">true</attribute>
      <rules>
        <contains element="activity" />
        <contains element="iteration" />
...
  <artifacts>
    <types/>
    <instances>
      <artifact type="Artifact">Project Plan</artifact>
...
  <tools/>
  <element type="sequence" name="OpenUP Process"
resource="6">
    <element type="phase" name="Inception"
milestone="Lifecycle Objectives">
      <element type="sequence">
        <element type="activity" name="Initiate Project">
          <element type="task" name="Develop Technical
Vision" resource="1">
            <output>
              <parameter name="vision"
tailoring="true">Vision</parameter>
              <parameter name="glossary"
tailoring="true">Glossary</parameter>
...
        <element type="parallel">
          <element type="activity" name="Identify and
Refine Requirements">
            <element type="sequence" resource="1">
...
                <element type="activity" name="Agree on
Technical Approach" resource="4">
...
      <element type="activity" name="Plan and Manage
Iteration" resource="2">
        <element type="sequence">
          <output>...
          </output>
```

Figure 10. Example OpenUP SEWL workflow snippets (end-tags omitted).

To retain the graphical details of the layout of nodes and edges, a separate file in XMI [39] notation was used. Figure 11 gives an example.

```
<graphicsystem:Graphicsystem
xmi:id='WUC_Phase_1_Inception'
parentDiagram='WUC_Process_OpenUPProcess.sewl_diagram' >
     <newObjects xmi:type='graphicsystem:Start'
xmi:id='startevent1' ObjectToObjects='sequenceStart1' />
  <newObjects xmi:type='graphicsystem:Sequenz'
xmi:id='sequenceStart1'
ObjectToObjects='WU_Activity_1_InitiateProject' />
   <newObjects xmi:type='graphicsystem:Activity'
xmi:id='WU_Activity_1_InitiateProject' Name='Initiate
Project'
Reference='WUC_Activity_1_InitiateProject.sewl_diagram'
ObjectToObjects='parallelGatewayStart1' />
...
 </graphicsystem:Graphicsystem>
 <notation:Diagram xmi:id='id_WUC_Phase_1_Inception'
type='SEWL' element='WUC_Phase_1_Inception'
name='Inception.sewl_diagram' measurementUnit='Pixel'>
    <children xmi:type='notation:Shape'
xmi:id='shape_startevent1' type='2043'
element='startevent1'>
...
   </children>
     <children xmi:type='notation:Node'
xmi:id='shape_WU_Activity_1_InitiateProject' type='2034'
element='WU_Activity_1_InitiateProject'>
     <children xmi:type='notation:DecorationNode'
xmi:id='4e841147-2f14-445a-b0b4-30e714be504e'
type='5039'/>
     <children xmi:type='notation:BasicCompartment'
xmi:id='0b62527e-b592-4e3d-a367-541f17843fb9'
type='7011'/>
     <styles xmi:type='notation:DescriptionStyle'
xmi:id='1b9fea72-5856-4be5-9203-1ef5cc58d000'/>
     <styles xmi:type='notation:FontStyle'
xmi:id='3051a516-b9f4-42c6-9698-8072fbe9a301'/>
     <styles xmi:type='notation:LineStyle'
xmi:id='7ea4d238-14fc-4068-a4ce-ed6bb08820af'/>
     <layoutConstraint xmi:type='notation:Bounds'
xmi:id='11135191-6e30-4c7a-a803-dfd437a058bc' x='440'
y='185' />
    </children>
...
 <styles xmi:type='notation:DiagramStyle'
xmi:id='_avAfkaznEeGl_a7M295XCw'/>
  <edges xmi:type='notation:Connector' xmi:id='flow23'
type='4020' source='shape_startevent1'
target='shape_sequenceStart1'>
   <styles xmi:type='notation:FontStyle'
xmi:id='8712763c-8e17-4285-948b-0b78f41f90af' />
     <element xsi:nil='true' />
     <bendpoints xmi:type='notation:RelativeBendpoints'
xmi:id='71805553-c9c1-46ff-8d13-56c6a3ab24fc'
points='[20, 0, -125, 10]$[130, -14, -15, -4]'/>
     <sourceAnchor xmi:type='notation:IdentityAnchor'
xmi:id='63f1b22c-d2fd-408e-9b8a-99044df18ce6' id='EAST'
/>
  <targetAnchor xmi:type='notation:IdentityAnchor'
xmi:id='0fd5db1f-daac-468a-a457-2dcf6bf1ee43'   />
   </edges>
```

Figure 11. Example SEWL diagram XMI code snippet.

An exemplary subset of the included constraints used to validate the model is listed here, i.e., audit rules. These were implemented in Java to allow usage outside of the GMF:

- verify phase/activity element has an output and a submodel,
- verify end element has no output,
- verify task does not target iteration/activity/phase,
- verify Loop has LoopEnd, Sequence has SequenceEnd, XOR has XOREnd, And has AndEnd.

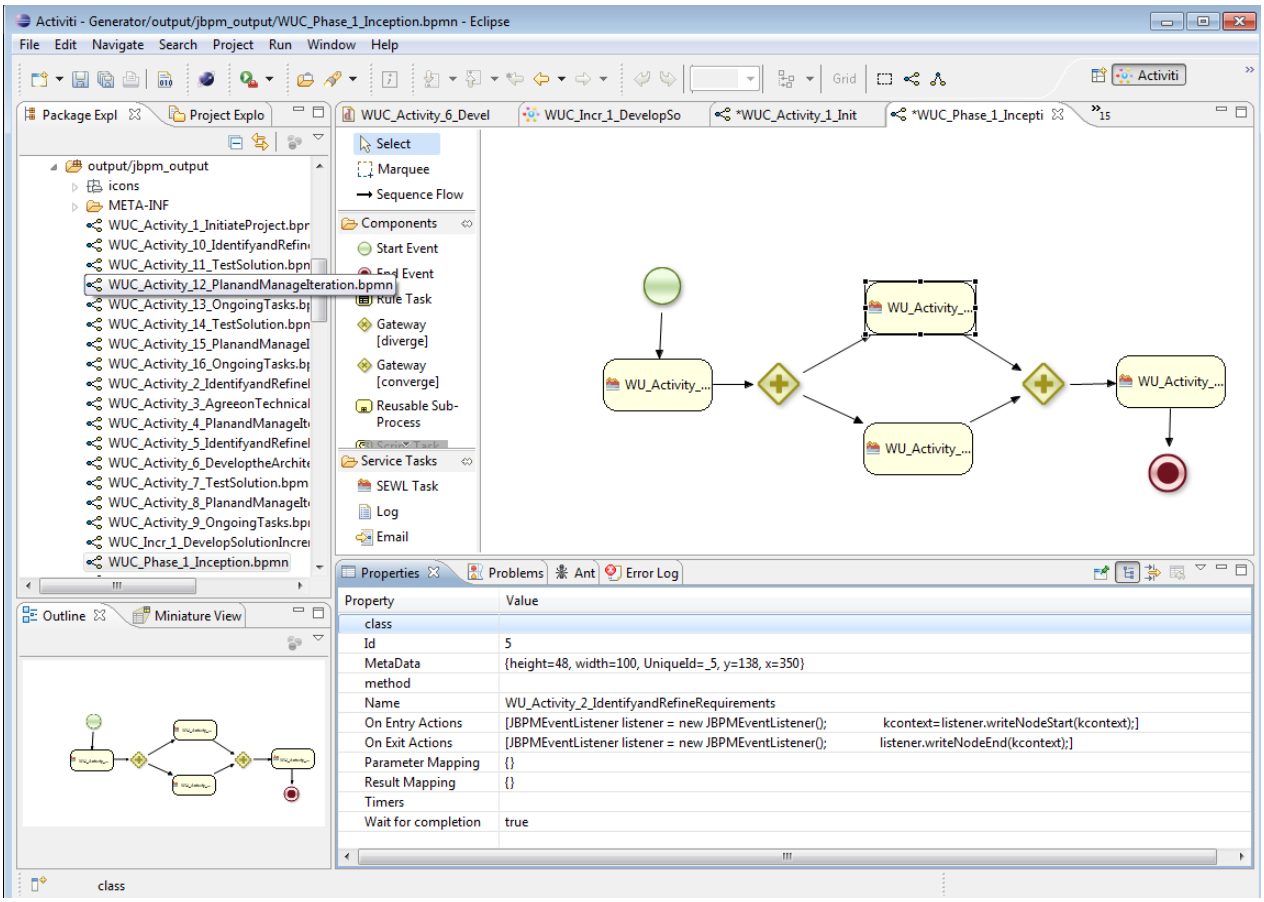


Figure 12. jBPM generated output (rearranged by hand).

V. EVALUATION

The evaluation of the solution method focuses on its practicality and viability. Three usage scenarios were evaluated, namely: a) the ability to use the method without utilizing any UMA model (SEWL only), b) starting with a new customized UMA model that represents an organization's own tailored SE process, and c) using an existing publicized UMA model. Furthermore, the performance of the method realization should be evaluated to determine if a model-driven XML-centric approach is adequate.

As to supporting a broad modeling spectrum, the Eclipse Process Framework (EPF) was used as a reference for modeling Scrum and OpenUP. These models were successfully modeled and transformed. Although the entire OpenUP process was modeled, only portions of the Inception Phase are shown below due to space constraints.

A. SEWL Non-UMA Process Model Example

The entire OpenUP process was modeled using the SEWL Editor as a starting point as was seen in Figure 9, and the generator was executed and jBPM and Activiti outputs were generated. Figure 12 was rearranged by hand. A snippet of the corresponding generated output is shown in Figure 14 for Activiti and in Figure 15 for jBPM. Thus processes that do not have or wish to use UMA can still utilize the method and SEWL for SE workflows.

B. New UMA Process Model Case Study

For this case, we started with a new EPF Composer project and modeled a portion of the Waterfall model based on Royce's original paper [40] as shown in Figure 13.

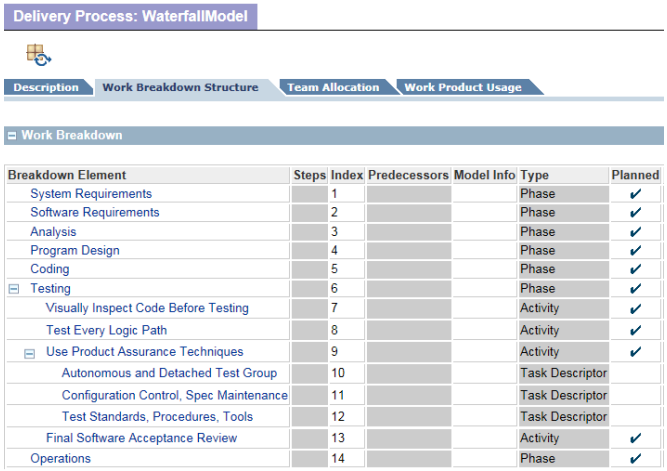


Figure 13. Screenshot of a Waterfall process modeled in EPF.

```
<<process id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
    <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.Proces
sStartEndListener'></activiti:executionListener>
   <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.Proces
sStartEndListener'></activiti:executionListener>
 </extensionElements>
  <startEvent id='startevent1' name='Start'></startEvent>
  <endEvent   id='endevent1'  name='End'></endEvent>
    <serviceTask id='WU_Activity_1_InitiateProject'
name='Initiate Project'
activiti:class='coseeek.workflow.process.activiti.extensi
on.DummyService '>
    <extensionElements>
     <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
     <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
    </extensionElements>
    </serviceTask>
   <parallelGateway id='parallelGatewayFork1' />
     <serviceTask
id='WU_Activity_2_IdentifyandRefineRequirements'
name='Identify and Refine Requirements'
activiti:class='coseeek.workflow.process.activiti.extensi
on.DummyService'>
      <extensionElements>
       <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
       <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
```

Figure 14. Example Activiti XML snippet.

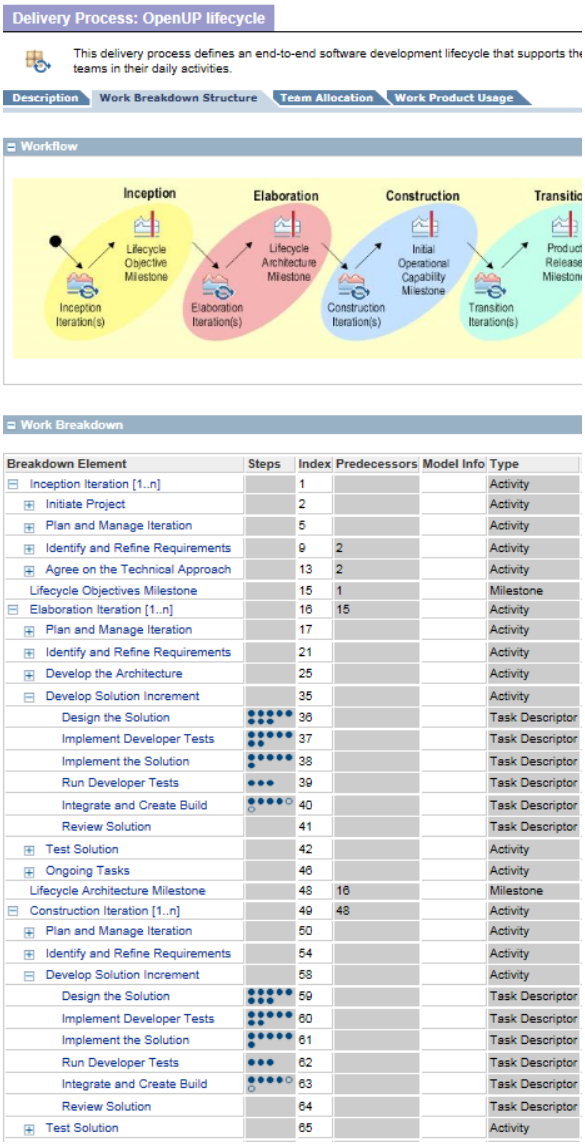
```
<process processType='Private' isExecutable='true'
id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
  <tns:import name='coseeek.workflow.process
.jbpm.extension.JBPMEventListener'/>
 </extensionElements>
  <startEvent id='_1' name='Start'></startEvent>
...
   <parallelGateway id='_3' gatewayDirection='Diverging'
/>
   <parallelGateway id='_4' gatewayDirection='Converging'
/>
  <task id='_5'
name='WU_Activity_2_IdentifyandRefineRequirements'
tns:taskName='SEWL Task' >
  <extensionElements>
  <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>JBPMEventListener listener =
      new JBPMEventListener();
   kcontext=listener.writeNodeStart(kcontext);</script>
  </tns:onEntry-script>
  <tns:onExit-script
     scriptFormat='http://www.java.com/java'>
    <script>JBPMEventListener listener = new
JBPMEventListener();
        listener.writeNodeEnd(kcontext);</script>
    </tns:onExit-script>
  </extensionElements>
  <ioSpecification>
    <inputSet/>
    <outputSet/>
  </ioSpecification>
  </task>
<task id='_6' name='WU_Activity_3_AgreeonTechnicalApp...
```

Figure 15. Example jBPM workflow snippet.

This demonstrates that an organization's model can be used conveyed to UMA, and that as long as phases, activities, and/or tasks are modeled, default sequential workflows can be automatically generated from this in SEWL as shown in Figure 17. Here, the *phases* are shown as a sequential workflow. The Testing Phase is shown as a workflow of *activities* as shown in Figure 18. In Figure 19, *tasks* within the activity related to product assurance techniques specified by the Waterfall model are shown. If desired, workflows can then modified in the SEWL, e.g., for more complex non-sequential workflow models. The SEWL workflows were automatically transformed by the Generator into corresponding jBPM workflows (shown in Figures 20-22) and Activiti workflows (shown in Figures 23-25).

C. *Existing UMA Process Model Case Study*

The published UMA process model OpenUP, shown in Figure 16, was modified to simulate a customization scenario for an organization. Here a "Review Solution" task was added to the "Develop Solution Increment" activity.



Breakdown Element	Steps	Index	Predecessors	Model Info	Type
Inception Iteration [1..n]		1			Activity
Initiate Project		2			Activity
Plan and Manage Iteration		5			Activity
Identify and Refine Requirements		9	2		Activity
Agree on the Technical Approach		13	2		Activity
Lifecycle Objectives Milestone		15	1		Milestone
Elaboration Iteration [1..n]		16	15		Activity
Plan and Manage Iteration		17			Activity
Identify and Refine Requirements		21			Activity
Develop the Architecture		25			Activity
Develop Solution Increment		35			Activity
Design the Solution		36			Task Descriptor
Implement Developer Tests		37			Task Descriptor
Implement the Solution		38			Task Descriptor
Run Developer Tests		39			Task Descriptor
Integrate and Create Build		40			Task Descriptor
Review Solution		41			Task Descriptor
Test Solution		42			Activity
Ongoing Tasks		46			Activity
Lifecycle Architecture Milestone		48	16		Milestone
Construction Iteration [1..n]		49	48		Activity
Plan and Manage Iteration		50			Activity
Identify and Refine Requirements		54			Activity
Develop Solution Increment		58			Activity
Design the Solution		59			Task Descriptor
Implement Developer Tests		60			Task Descriptor
Implement the Solution		61			Task Descriptor
Run Developer Tests		62			Task Descriptor
Integrate and Create Build		63			Task Descriptor
Review Solution		64			Task Descriptor
Test Solution		65			Activity

Figure 16. OpenUP screenshot showing Review Solution customization.

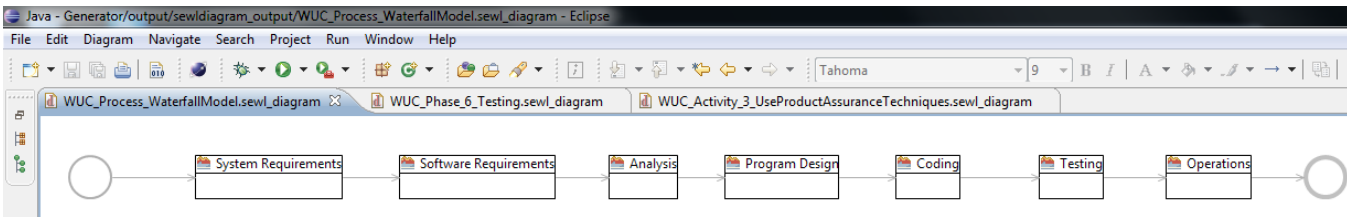


Figure 17. SEWL diagram of the Waterfall phases.

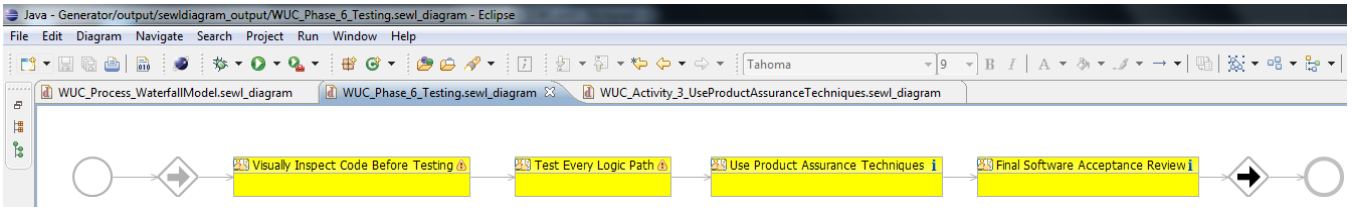


Figure 18. SEWL workflow diagram of Waterfall's Testing Phase.

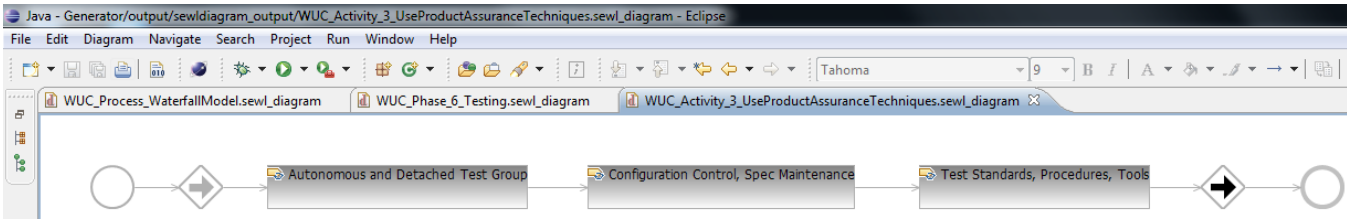


Figure 19. SEWL workflow diagram showing tasks of the Waterfall's activity Use Product Assurance Techniques.

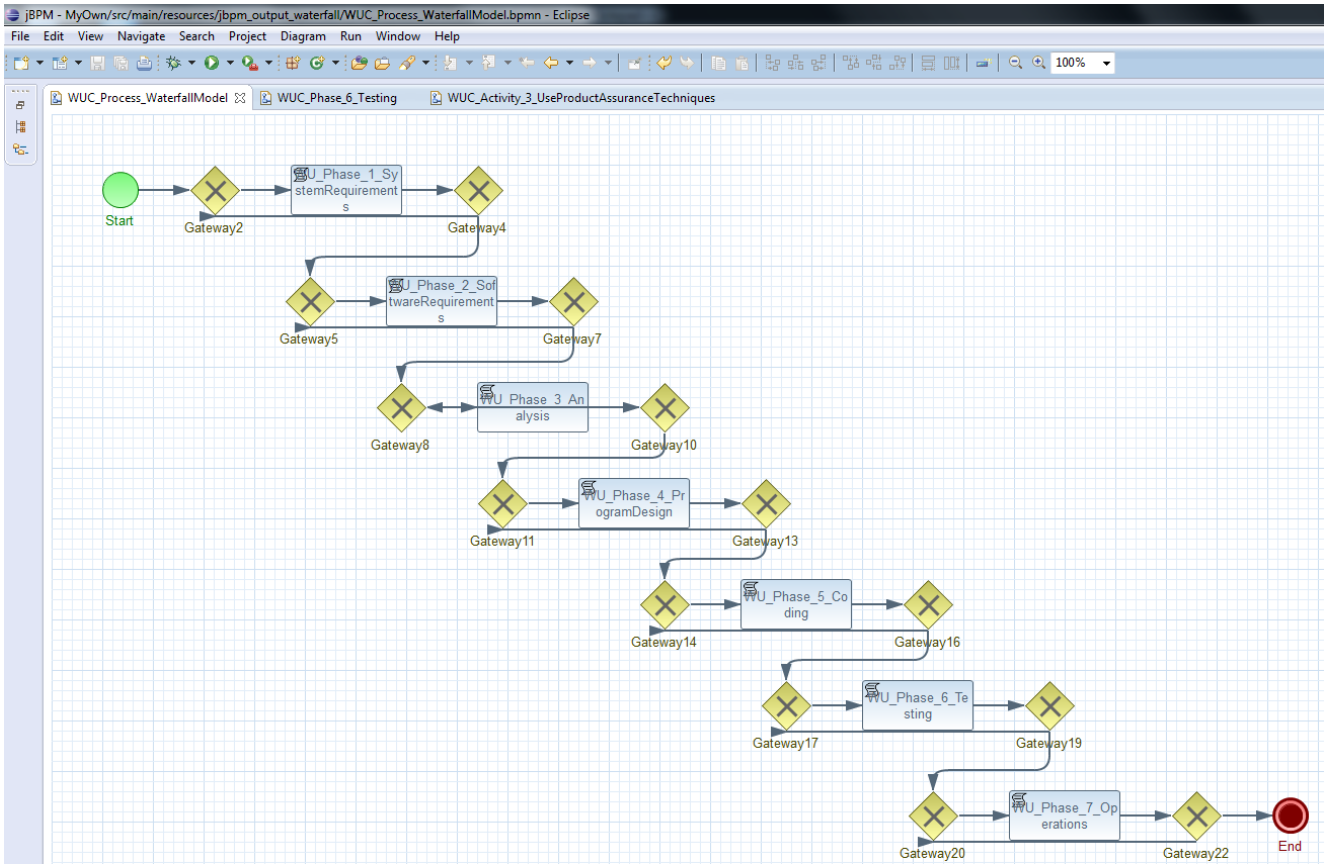


Figure 20. jBPM diagram of the Waterfall phases workflow.

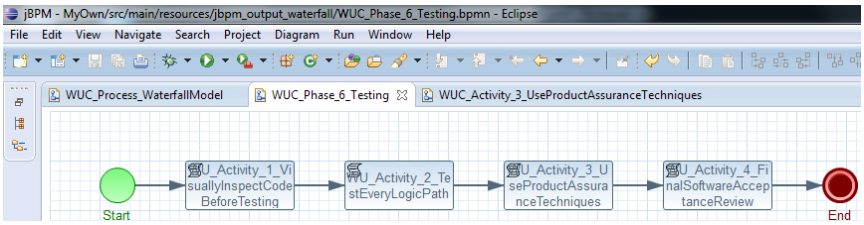


Figure 21. jBPM diagram of Waterfall's Testing Phase Workflow.

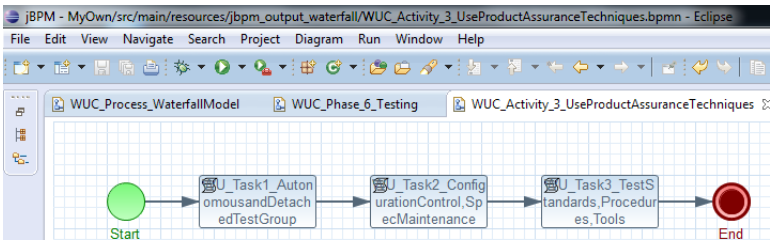


Figure 22. jBPM diagram of Waterfall's activity Use Product Assurance Techniques.

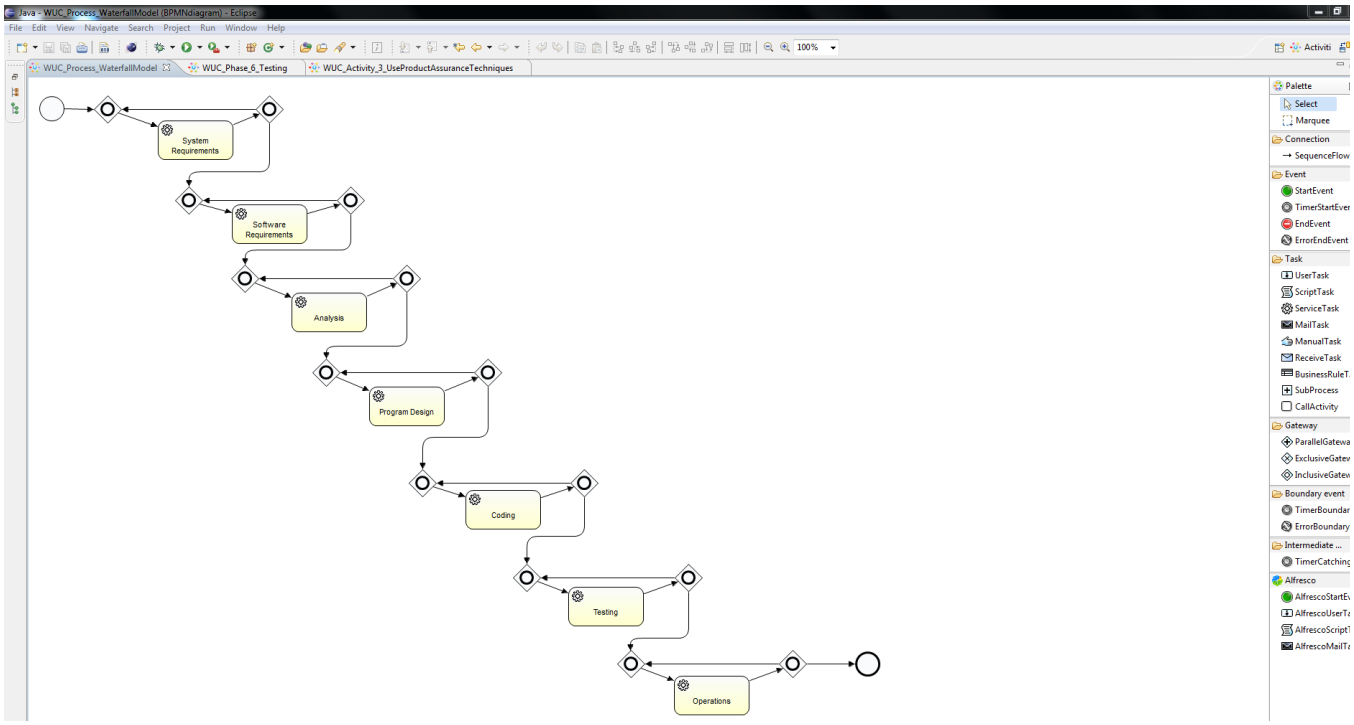


Figure 23. Activiti diagram of Waterfall's phases workflow.

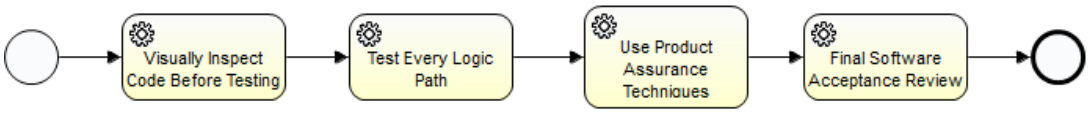


Figure 24. Activiti diagram of Waterfall's Test Phase workflow.

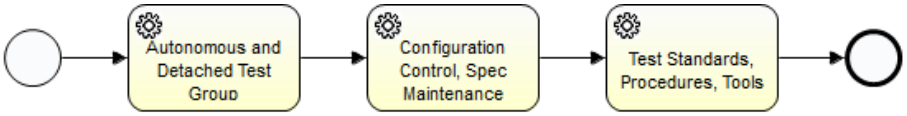


Figure 25. Activiti diagram of the tasks in the Waterfall's activity Use Product Assurance Techniques.

This was done to demonstrate that an existing comprehensive UMA model from the community can be customized and default sequential workflows automatically generated in SEWL. In Figure 26, the phases are shown as a sequential workflow. The Construction Phase was modeled as a workflow of activities in Figure 27. In Figure 28, the tasks for the activity related to product assurance techniques are shown. If necessary, these workflows can then be modified in the SEWL graphical editor to suit the needs for more complex non-sequential workflow models. Nevertheless, a starting basis is automatically provided. From the SEWL model, we transformed these SEWL workflows using the Generator into corresponding jBPM (Figures 29-31) and Activiti (Figures 32-34) workflows.

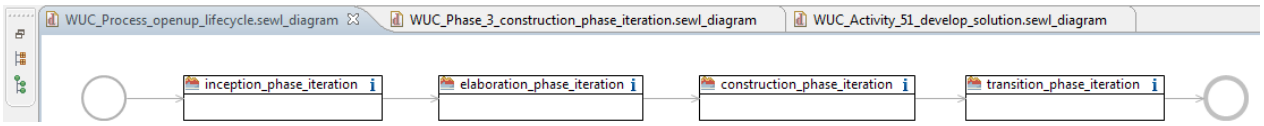


Figure 26. SEWL diagram of the OpenUP phases workflow.

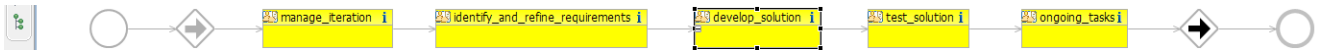


Figure 27. SEWL diagram of OpenUP's Construction Phase workflow.

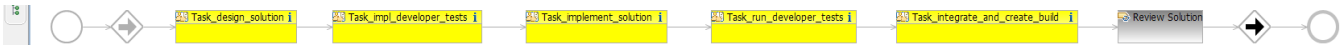


Figure 28. SEWL diagram of the OpenUP's Develop Solution Increment workflow.

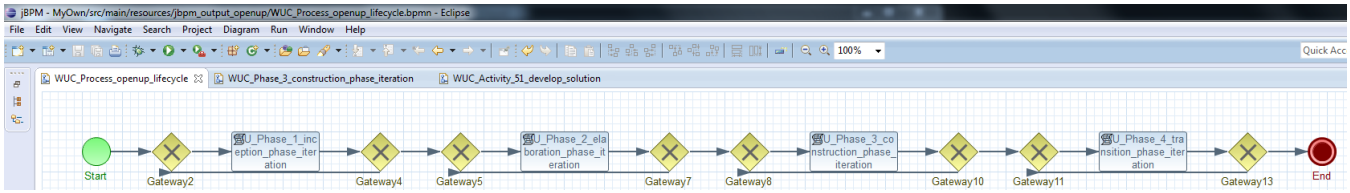


Figure 29. jBPM diagram of the OpenUP phases workflow.

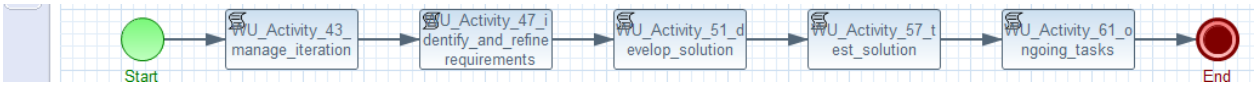


Figure 30. jBPM diagram of OpenUP's Construction Phase workflow.

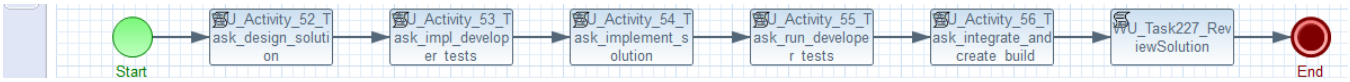


Figure 31. jBPM diagram of the OpenUP Develop Solution Increment workflow.

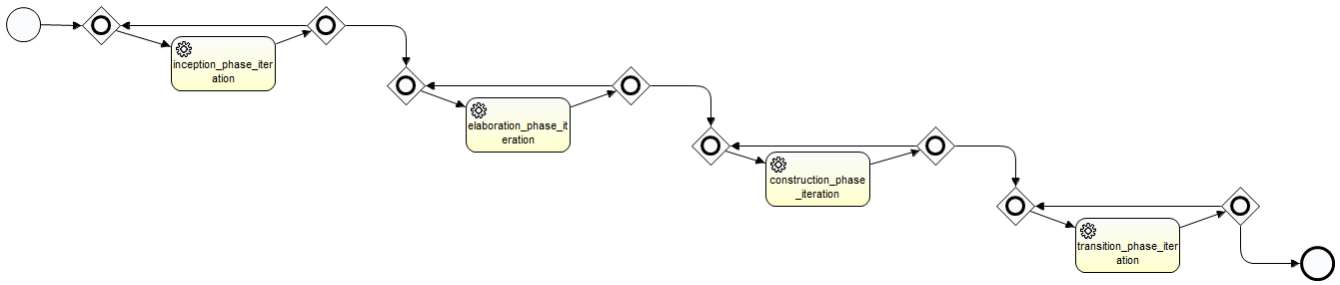


Figure 32. Activiti diagram of the OpenUP phases workflow.

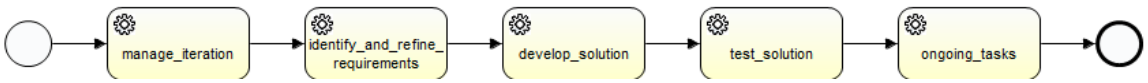


Figure 33. Activiti diagram of OpenUP's Construction Phase workflow.

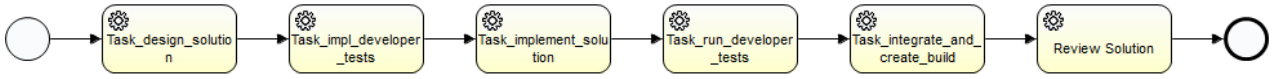


Figure 34. Activiti diagram of OpenUP's Develop Solution Increment workflow.

D. Performance

We evaluated our model-driven solution to determine if it exhibits acceptable transformation performance for expected usage.

1) Performance for provided SE Models: For this scenario we wanted to determine the performance one can expect for model transformation of basic and realistic SE process models that conform to the EPF XML Schema. The configuration for these measurements consisted of an Intel Core i7-3740QM CPU @2,70GHz, 16 GB RAM, Windows 7 Ultimate SP1 x64, Java 8, Scala 2.11.5.

The performance results presented in Table II differentiate the two UMA SE processes Waterfall and OpenUP, with Waterfall being a new relatively small model example, and OpenUP being a comprehensive model example. The second major column shows the resulting file sizes in bytes, lines, and number of files involved for each of the generation steps. The major center column then shows the duration in milliseconds for various transformations. The EPF input files are the starting point, with the generation steps being the EPF to SEWL transformation (EPF->SEWL), the SEWL to diagram transformation (SEWL->diagram), the SEWL to jBPM transformation (SEWL->jBPM), and the SEWL to Activiti transformation (SEWL->Activiti). The maximum time needed was about 1 second needed to create 83 XMI diagram files with over 10,0000 XML lines from the input of 995 lines of SEWL XML. This performance seems acceptable for such a comprehensive SE model.

TABLE II. EPF MODEL TRANSFORMATION PERFORMANCE

Generation Steps	Duration (milliseconds)		Generated XML in bytes [lines] (files)	
	Waterfall	*OpenUP*	*Waterfall*	*OpenUP*
EPF Input files	-	-	9,480 [75] (1)	1,671,816 [24109] (1)
EPF -> SEWL	341	505	3,084 [85] (1)	51,695 [995] (1)
SEWL -> diagram	811	1040	32,344 [332] (3)	1,008,362 [10277] (83)
SEWL -> jBPM	646	833	20,929 [442] (12)	222,632 [5438] (83)
SEWL -> Activiti	609	950	17,218 [183] (3)	383,087 [3747] (83)

2) Plugin Performance: We compared performance of the various generator plugins for their different types of output to determine if there are significant differences or issues with such a plugin concept. For this, a five node OpenUP process sequence was provided as the input to the SEWL Editor, and the performance of each of the adapters in the Generator measured. For each round, a loop of 1000 generations was averaged. For generating the SEWL template, the SEWL diagram files served as input. Otherwise the SEWL XML template alone was used as input. The configuration for these measurements consisted of an Intel Core 2 Duo CPU 2.26 GHz, 3 GB RAM, Windows XP Pro SP3, Java JDK 1.6.0-31, Scala 2.9.1, Activiti 5.8, jBPM 5.2, Jena 2.6.4, and Eclipse EMT (Helios) SR2. The performance results are presented in Table III, with the left column indicating the adapter measured, the center column the average duration, and the right column the size of the inputs and outputs in bytes, lines, and files.

TABLE III. GENERATOR ADAPTER PERFORMANCE

	Average Duration (millisec)	XML File Size in bytes [lines] (files)	
		Input	*Generated*
SEWL template	10.3	212,490 [2255] (22)	19,431 [416] (1)
SEWL-Diagram	65.1	19,431 [416] (1)	212,490 [2255] (22)
Activiti	23.8	19,431 [416] (1)	79,088 [822] (22)
jBPM	27.5	19,431 [416] (1)	86,169 [1856] (22)
Ontology	6917.8	19,431 [416] (1) 823,020 [12965] (1)	1,469,639 [15750] (1)

Generating a SEWL template from the diagram involves the least amount of writing, and is thus fastest. The generation of SEWL diagrams in XMI format is more verbose in bytes and lines by at least a factor of 2, and its duration is correspondingly longer compared to the jBPM or Activiti adapters. With regard to the Ontology adapter, two files serve as input for generating the OWL ontology; in addition to the SEWL template input, the Jena Semantic Web framework is used to parse and create internal objects from the existing ontology (a comparatively large file with its additional 12 related remote namespace schema), then the relevant ontology instances are updated based on the SEWL file, and finally a complete OWL file that contains the modifications is generated. Since much more XML is involved in both the input parsing and generation, and the use of specialized semantic OWL APIs, here the overall performance for generating semantic workflow context concepts is noticeable.

Because the RDF and OWL-DL XML formats are standardized, possible optimization strategies include partitioning the ontology to only those areas applicable for workflow ontology concepts, rather than the total ontology. Another possibility is the use of solid state disks on the devices involved in the ontology generation, e.g., by placing the adapter behind a web service.

In summary, the performance of the generators appears satisfactory for typical SE process transformation.

VI. Discussion

While there is potential for further automation of SE processes and automated guidance support, a number of practical hindrances remain.

In the past, since SE process documentation lacked contextually adapted and WfMS supported workflows, it has often seemed not to be operationally relevant, but rather something relatively abstract. It might serve to satisfy the appearance of the existence of a disciplined and professional method for approaching the software engineering work, with no real way to monitor actual usage or compliance with it. Thus, for most of the actors involved in using the documentation of SE processes, the documented workflows appear not to add significant value and most of the work is done without referring to the SE process documentation with any of its specified abstract workflows. Perhaps the swing in prior decades to overly documented und irrelevant SE processes caused an understandable and reactionary agile movement, as codified in the Agile Manifesto [1], to minimize tools and documentation.

However, if SE process documentation could include operationally concrete and WfMS-enactable workflows that provide contextually relevant guidance, and these workflows involved the actual tools and artifacts used and were tied in to the SE process documentation as well, then it would bring "life" to the relatively "dead" SE process documentation, since contextually adapted workflows would be mostly relevant and helpful to the software engineers in their actual work context. They would no longer have the hurdle of manually finding their current context in the abstract and perhaps quite comprehensive SE process documentation, and then manually determining the workflow portions they are supposed to follow and keep jumping from their work context to their process documentation context.

Activities and processes are an inherent part of SE and will continue to be used to accomplish SE work, be they explicit and documented or implicit and undocumented. As automation and intelligence in SE environments increases, a middle-ground may be found between these two extremes, so that the benefits that other domains have reaped from WfMS support for human-centric and hybrid human and automated workflows can be better integrated and leveraged in the SE landscape. This will involve documentation and workflow modeling, and a model-driven approach can automate many of the aspects in order to minimize the effort involved for software engineers.

The SE environment, tooling, and process area has seen relatively little standardization for various reasons. Efforts to better integrate the heterogeneous SE tooling landscape in a vendor-neutral manner, such as the semantic services approach of Open Services for Lifecycle Collaboration (OSLC) project, seem to have fizzled at the moment for all practical purposes. As tools continue to change, keeping the data models and integrations updated seems to involve significant effort without significant incentives. Future approaches may move more tooling to the cloud, enabling better context-aware SE integration.

VII. Conclusion and Future Work

This article contributed a practical model-driven methodology that supports the usage and transformation of software engineering process documentation to workflows executable in modern workflow management systems. The method can utilize comprehensive SE process documentation meta-models and automatically extract incorporated SE concepts and workflow models to an intermediate SE workflow format such as the Software Engineering Workflow Language (SEWL). These workflows can optionally be edited in a common SE format that is aware of SE process and environment concepts and can then transformed into various enactable WfMS formats and ontological concepts for context-aware support.

Automated workflow guidance for SE projects remains a challenge, and current SE process meta-models have hitherto not integrated support for intricate and enactable workflow modeling capabilities. With our practical model-driven method we showed that, beginning with only a rudimentary process documentation of a set of SE concepts conformant to some SE process meta-model such as the UMA, actually enactable workflows can be automatically generated for various common WfMS. Our method enables the utilization of currently available SE process documentation tooling such as the EPF Composer, without needing to deal with separate manual process modeling techniques for a vendor-specific WfMS due to our model-driven adapters. Such generated sequential workflows extracted and transformed from some SE process documentation can provide a starting point for more intricate operational SE workflow modeling in, for instance, our SEWL editor, should certain workflows be more complex or require branches or loops. These can be readily adjusted either with the SEWL graphical editor or directly in the SEWL text-based model, and then WfMS-specific workflows can be automatically generated.

This solution method provides an easy to use graphical modeling capability for executable SE workflows that can execute on commonly available WfMS, while retaining SE semantic information in a separate OWL file for contextually aware PCSEEs. The evaluation results show that such a model-based method for transforming SE workflows to common WfMS is both feasible and practical.

Future work includes case studies with industry partners in live settings as well as more comprehensive utilization of the ontological concepts extractable from such UMA-based models with those of CoSEEEK. Also, bidirectional workflow transformation support between SEWL and an engine-specific workflow format would allow editing in the workflow editor of choice. This entails providing reverse transformation support for engine-specific workflow templates, enabling engine-specific usage of features and editing capabilities via workflow engine-specific editors. For instance, changes made to jBPM and Activiti workflows could be automatically reflected in a SEWL template.

ACKNOWLEDGMENT

The author thanks and acknowledges Vitali Koschewoi and Julian Donauer for their work on the implementation and diagrams, and Gregor Grambow for his assistance with CoSEEEK-related concepts, ontology, and adaptations.

REFERENCES

[1] R. Oberhauser, "An Approach for Modeling and Transforming Contextually-Aware Software Engineering Workflows," Proceedings of the Ninth International Conference on Software Engineering Advances (ICSEA 2014), published by IARIA, ISBN: 978-1-61208-367-4 (2014), pp. 117-122.

[2] S. Biffl, D. Winkler, R. Höhn, and H. Wetzel, "Software process improvement in Europe: potential of the new V-modell XT and research issues," Software Process: Improvement and Practice, 11(3), 2006, pp. 229-238.

[3] P. Kroll and B. MacIsaac, Agility and Discipline Made Easy: Practices from OpenUP and RUP. Pearson Education, 2006.

[4] http://v-modell.iabg.de/v-modell-xt-html-english/ 2015.05.30.

[5] http://epf.eclipse.org/wikis/openup/ 2015.05.30.

[6] http://www.eclipse.org/epf/ 2015.05.30.

[7] P. Haumer, "Eclipse process framework composer," Eclipse Foundation, 2007.

[8] Object Management Group, "Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0," Object Management Group, 2008.

[9] V. Gruhn, "Process-Centered Software Engineering Environments: A Brief History and Future Challenges," Annals of Software Engineering, 14(1-4), 2002, pp. 363-382.

[10] A. Fuggetta, "Software process: a roadmap," Proc. Conf. on the Future of Software Eng., ACM, May 2000, pp. 25-34.

[11] M. Reichert and B. Weber, "Enabling flexibility in process-aware information systems: challenges, methods, technologies," Springer Science & Business Media, 2012.

[12] R. Oberhauser, "Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments," Semantic Web, Gang Wu (ed.), In-Tech, Austria, 2010.

[13] G. Grambow, R. Oberhauser, and M. Reichert, "Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects," Software and Data Technologies (Editors: J. Cordeiro, M. Virvou, B. Shishkov), CCIS 303, Springer Verlag, ISBN 978-3-642-29577-5, 2012, pp. 73-88.

[14] G. Grambow, R. Oberhauser, and M. Reichert, "Contextually Injecting Quality Measures into Software Engineering Processes," the International Journal On Advances in Software, ISSN 1942-2628, vol. 4, no. 1 & 2, 2011, pp. 76-99.

[15] Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," 2011.

[16] G. Grambow, R. Oberhauser, and M. Reichert, "Towards a Workflow Language for Software Engineering," Proc. of the The Tenth IASTED Int'l Conf. on Software Engineering (SE 2011), ISBN 978-0-88986-880-9, ACTA Press, 2011.

[17] P. Dadam et al., "From ADEPT to AristaFlow BPM suite: a research vision has become reality," in Business process management workshops, Springer, Jan. 2010, pp. 529-531.

[18] W. Van Der Aalst and A. Ter Hofstede, "YAWL: yet another workflow language," Information systems, 30(4), 2005, pp. 245-275.

[19] M. Salatino and E. Aliverti, jBPM5 Developer Guide, ISBN 1849516448, Packt Publishing, 2012.

[20] T. Rademakers, "Activiti in Action: Executable business processes in BPMN 2.0," Manning Publications Co., 2012.

[21] R. Bendraou, B. Combemale, X. Crégut, and M. Gervais, "Definition of an Executable SPEM 2.0," In Proc. APSEC 2007, IEEE, 2007, pp. 390-397.

[22] N. Debnath, D. Riesco, M. Cota, J. Garcia Perez-Schofield, and D. Uva, "Supporting the SPEM with a UML Extended Workflow Metamodel," Proc. IEEE Conf. on Computer Systems and Applications, AICCSA, 2006, pp. 1151-1154.

[23] D. Riesco, G. Montejano, N. Debnath, and M. Cota, "Formalizing the Management Automation with Workflow of Software Development Process Based on the SPEM Activities View," Proc. 6th Int'l Conf. on information Technology: New Generations, 2009, pp. 131-136.

[24] M. Perez Cota, D. Riesco, I. Lee, N. Debnath, and G. Montejano, "Transformations from SPEM work sequences to BPMN sequence flows for the automation of software development process," J. Comp. Methods in Sci. and Eng. 10, 1-2S1, (September 2010), 2010, pp. 61-72.

[25] Y. Feng, L. Mingshu, and W. Zhigang, "SPEM2XPDL: Towards SPEM Model Enactment," Proc. of SERP, 2006, pp. 240-245.

[26] C. Portela et al. "xSPIDER ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0," J. of SW Eng. & Applications, 5(6), 2012, pp. 375-384.

[27] D. McGuinness and F. Van Harmelen, "OWL web ontology language overview," W3C recommendation, 2004.

[28] D. Gasevic, D. Djuric, and V. Devedzic, Model driven architecture and ontology development. Springer, 2006.

[29] D. Gelernter, "Generative communication in Linda," ACM Transactions on Programming Languages and Systems, 7(1), 1985, pp. 80-112.

[30] W. Meier, "eXist: An open source native XML database. Web," Web-Services, and Database Systems, LNCS, 2593, 2009, pp. 169-183.

[31] http://www.eclipse.org/modeling/gmp/ 2015.05.30.

[32] http://www.eclipse.org/modeling/emf/ 2015.05.30.

[33] B. McBride, "Jena: a semantic web toolkit," Internet Computing, Nov. 2002, pp. 55-59.

[34] N. F. Noy et al., "Creating semantic web contents with protege-2000," IEEE intelligent systems, 16(2), pp. 60-71, 2001.

[35] G. Grambow, R. Oberhauser, and M. Reichert, "User-centric Abstraction of Workflow Logic Applied to Software Engineering Processes," Proceedings of the 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), 2012.

[36] G. Grambow and R. Oberhauser, "Towards Automated Context-Aware Selection of Software Quality Measures," Proc. 5th Intl. Conf. on Software Engineering Advances, 2010, pp. 347-352.

[37] G. Grambow, R. Oberhauser, and M. Reichert, "Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management," Proc. 2nd Int'l. Conf. on Adaptive and Self-adaptive Systems and Applications, 2010, pp. 58-67.

[38] G. Grambow, R. Oberhauser, and M. Reichert, "Contextual Quality Measure Integration into Software Engineering Processes," International Journal on Advances in Software, 4(1&2), 2011, pp. 76-99.

[39] Object Management Group, "MOF 2 XMI Mapping Version 2.4," 2010.

[40] W. W. Royce, "Managing the development of large software systems," Proceedings of IEEE WESCON, Vol. 26, No. 8, 1970, pp. 328-339.

[41] K. Beck et al., "The agile manifesto," 2001.

Modeling Responsibilities of Graphical User Interface Architectures via Software Categories

Stefan Wendler
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
stefan.wendler@tu-ilmenau.de

Abstract — **Business information of our days systems heavily rely on graphical user interfaces (GUIs) as a sub-system that provides rich interaction options to access business services and stands out with high usability. To develop and maintain a GUI sub-system, high efforts accumulate due to missing standard solutions and limited reuse of already established architectures. Published architectural patterns and few reference architectures are primary sources for GUI architecture development. However, these concepts need to be extensively adapted, since individual requirements are to be met and available sources do not describe all necessary details. These are fine-grained GUI responsibilities, differentiated state handling for application and presentation as well as implementation structures. Therefore, GUI development projects create high efforts and their resulting architecture often does not represent the desired separation of concerns, and so, is hard to maintain. These architectures are no proper foundation for the integration of recent user interface pattern (UIP) concepts, which promise a reuse of proven usability concepts and enable the automated generation of vast GUI parts. In this work, the design issues that occur during GUI architecture development are to be analyzed. To prepare the analysis, selected GUI architecture and pattern concepts are presented. Furthermore, the general responsibilities of GUI sub-systems and their structural elements are identified. In detail, software categories are applied to model the GUI responsibilities and their relationships by separating their concerns based on several dimensions of knowledge. The resulting software category tree serves as a basis to review the well-known model view controller pattern and the Quasar client architecture, which is a detailed GUI reference architecture of the domain. As result, the major design issues of GUI systems are derived and summarized. Eventually, the created GUI software category tree can be applied as a foundation for the creation, understanding and assessment of other GUI patterns or reference architectures.**

Keywords — GUI software architecture; software architecture; user interface patterns; graphical user interface.

I. INTRODUCTION

A. Motivation

Domain. Business information systems represent a domain that is largely influenced by software architecture considerations. Especially the graphical user interface (GUI) sub-system is likely to induce high efforts [2] for both development and later maintenance. According to a survey among 23 major Germany-based IT service companies, IT departments of banking, logistics and power supply industries, as well as medium-sized IT developers, the development efforts for GUI systems is still estimated to be considerably high compared to other common sub-systems of business information systems. On a basis of 100% total development effort, the aggregated efforts across all development phases were estimated for the four principal business information systems layers as follows: workflow layer 24,8%; presentation layer or GUI 26,8%; application layer 29,8% and lastly persistence layer 18,6%. In sum, the presentation layer was rated as the second highest concerning total effort.

The high efforts for GUI development apply for both standard and individual software systems as a high demand for individually designed GUI systems is actually present. The companies require their business information systems to be closely matched to their business processes. As a consequence, custom services are often to be developed or configured, which require a customized GUI to reflect the functional aspects. In addition, a high usability is almost always a desired goal to achieve during the development of new GUI dialogs.

Problems. However, GUI architectures are not standardized to the required detail, since historically applied patterns have not converged towards a detailed standard architecture that models every responsibility needed for considering current functional, usability and technological influences during development or maintenance. According to functional aspects, the higher degree of system integration into business processes demands for exact implementations of comprehensive requirement artifact types like use cases, tasks and business processes. The customers expect the GUI system to closely match the specified scenarios with dialogs that reflect the flow and branching of actions along with the proper display of context relevant and even optional data. Users no longer reenact those scenarios by activating the single functions with their belonging dialogs in the right order. They expect the GUI system to provide guidance instead, navigation facilities and adequate presentation layouts to attain a dialog structure that perfectly mirrors and complements the functional requirements specification.

Those current GUI development needs are facing rather old GUI architecture patterns like model view controller (MVC) [3] and its variants [4], which did not consider such a deep and vast requirements basis. To resolve some MVC limitations or add some detail, other MVC pattern derivates like HMVC [5], MVP [6][7], MVVM [8] and MVA, RMR, ADR (reference [9] provides some overview only) were

This work is an extensively revised and substantially augmented version of "A Software Category Model for Graphical User Interface Architectures", which appeared in the Proceedings of The Ninth International Conference on Software Engineering Advances (ICSEA 2014) [1].

introduced and occasionally found their role as a principal architecture of GUI frameworks. However, mostly single dialogs are considered by those patterns, so that concerns like the design of navigation among dialogs, the structuring and separation of visual layout, presentation control, dialog control [10] and application flow are not comprehensively described by a single pattern. There is no standard solution available by the books; many sources [11][12][13] focus on the proper handling of programming languages or mastering certain GUI framework facilities without paying much attention to architecture structuring. Thus, many details remain to be refined by the developers [14], who will adapt architectures individually for each system and most likely will not extract a commonly reusable architecture due to lacking time or budget.

According to Fowler [15], during the course of analyzing and refining patterns many different interpretations may emerge, so that there will be no common understanding of a single pattern and its involved roles. This may due to the complex structure of patterns, which are regularly containing several ideas at once that may even comprise smaller sub-patterns. Thus, developers will instantiate one pattern according to their gained understanding, experience with other patterns and the integration of surrounding frameworks and architecture aspects to be addressed.

Ultimately, there is no consensus on GUI patterns, which one offers the optimal structuring of responsibilities, so that it is fairly common to decide on their application and adaptation anew for each GUI development project. Although MVC is very commonly applied, this pattern also is very often misunderstood [15]. To apply common GUI architecture patterns in practice, several implementation problems have to be solved that are not sufficiently addressed by the patterns [10]. Besides, reference architectures [2][16] and several patterns (design and architectural) [17][18] had been suggested, but have not been properly integrated with traceability [19][20] concepts to keep track of requirements during architecture design.

Moreover, GUI frameworks often dictate to closely adopt a certain pattern-based architecture, so that they have a large impact on the GUI system's structure and often cannot be isolated properly to separate technical implementations from domain or project specific requirements.

So far, the functional aspects were considered. As far as the demand for high usability is concerned, the above mentioned patterns do not solve the integration of ergonomically effective presentation layouts or interaction designs. They focus on mere technical reuse of software architecture structure and do not consider content-based reuse.

Consequences. Foremost, GUI systems remain hard to develop concerning the effective adaptation of available patterns or reference architectures, as well as the cost-efficient implementation of current functional and usability requirements. In addition, developers may be frequently required to work with a certain GUI framework to be able to integrate the new created GUI system parts with an existing system or maintain a certain corporate design already in use with other neighboring systems. In the end, the coupling between system layers and the separation of concerns remain vague due to different pattern characteristics and project budgets.

Furthermore, when systems have grown after several maintenance steps, different concerns tend to be mixed up within the GUI architecture the larger the requirements basis is and the more complicated the integrated frameworks are. For instance, application server calls, data handling, task and dialog control flow can no longer clearly allocated to certain elements of the software architecture. These concerns are likely to be scattered among several units of design. Finally, the GUI and application sub-systems cannot be separated easily, so that the evolution of both depends on each other. Business logic tends to be scattered in the GUI dialogs [21] and the "smart UI antipattern" [22] may become a regular problem. Initially, the architecture was layered during design phase, but the encapsulation of components and separation of concerns did not prove in practice [21]. This is maybe due to used frameworks that expect a certain architecture, which alters original design. More likely is the phenomenon that the architecture was based on common patterns and reference architectures that could not be refined in time with respect to desired quality and extensibility. Lastly, the two concluding points from Siedersleben [21] are still of relevance: standardized interfaces between layers are missing and technical frameworks dominate the architecture and evolution. Currently, there are often more than three layers in business information systems and the segregation got even more complex.

User interface patterns. There are perspectives that are promising to address the persisting issues. Current research is occupied with the integration of a new artifact type in the development of GUI systems. Being based on design pattern concepts and likewise description schemes [23], user interface patterns (UIPs) have been approached [24][25][26] to facilitate the generative development of GUIs and highly increase the reuse of proven visual and interaction design solutions that originate from descriptive human computer interaction (HCI) patterns [27][28].

According to the generative nature of UIP integration approaches, the development of GUIs shall be shortened by model-based sources that specify both the GUI system's view instances and the coupling between functional related and GUI system architecture components. This new kind of pattern is intended to bridge the gap between descriptive HCI patterns and implementation oriented architecture patterns. Ultimately, with the application of UIPs the technical reuse of architecture structures of common design or architecture patterns shall be combined with the reuse of content relevant for ergonomics (visual design and layout, interaction design, HCI patterns) bound to certain design units, which usually remain abstract in common pattern descriptions. In that way, UIPs shall be stored in a repository to be configured and instantiated for different projects. In short, both technical architecture parts and visual design shall be coupled and reused in different contexts.

Current limitations. Currently, there are still design issues within GUI patterns or reference architectures that hinder the evolution and maintenance of existing systems. To establish a target software architecture of high quality for the implementation of UIPs, these issues have to be addressed in

the first place. A commonly applicable GUI architecture has to be derived. In fact, UIPs need a clear basis of reuse: an architecture with well separated concerns that permits the flexible allocation and exchange of these greater units of design without the need to adapt other components. Otherwise, the previously described problems would persist: due to lacking standard solutions, each project would need an individual target architecture with every responsibility detailed to accept UIP instances. Generator templates would have to be created and revised over and over again for any GUI framework or platform. The visual designs of UIPs would only be available for specification and context configuration, but would miss a technical architecture for their implementation on a certain system. To be able to increase reuse with UIPs, a standardized architecture solution is truly needed. The individual refinement of patterns will greatly hinder the benefits UIP-based reuse would promise.

Whether UIPs will be generated, interpreted or provided by a virtual user interface [29][30] the resulting architecture will be at least as complex as for standard GUIs. Therefore, the common issues in design will prevail and affect UIP based solutions.

B. Objectives

To prepare the integration of UIPs into GUI architecture and at the same time preserve their reusability and variability in different contexts, open issues in GUI architecture development have to be identified and solved. Therefore, our first objective is to provide a detailed analysis of perpetual design problems. Design issues regularly occur whenever one of the following cases is encountered:

- Requirements are not met due to missing allocation of responsibilities to design units.
- Several design units share are certain set of responsibilities, so that either cohesion or separation of concerns is not ideal.
- One design unit takes responsibility for many tasks at once, and thus, may not represent a proper degree of cohesion.

Hence, we will have to identify the re-occurring responsibilities of GUI architectures to be able to analyze possible GUI design issues. In this regard, our second objective is to derive a pattern- and architecture-independent model of those responsibilities and their relationships.

On that basis, the frequently applied MVC pattern is reviewed. In addition, we will analyze the Quasar client reference architecture [2] that provides more detail than regular patterns and was created especially for the domain. Together with the presentation of selected related work, the responsibilities model and the analysis will lead us to reveal persisting issues in GUI design.

C. Structure of the Paper

The following section provides descriptions of common patterns and reference architecture considerations for GUIs of our particular domain. In the third section, we will elaborate a general responsibilities model for GUI architectures. In Section IV, the GUI architecture patterns are reviewed. The results are summarized and discussed in Section V, before we conclude in Section VI.

II. Related Work

A. Architecture Patterns for Graphical User Interfaces

With the invention of object oriented programming languages, a clear assignment of the cross-cutting concerns, which are common for a GUI dialog, had to be enforced.

Eventually, the MVC pattern was introduced [3], which distinguishes three object types as abstractions to accept defined responsibilities. The typical roles of an MVC triad are the following: *View* and *Controller* comprise the GUI part; the *Model* represents the application parts with the core functionality and data structures [18].

With these roles, the MVC pattern promised a separation of concerns, modularity and even reuse of selected abstractions [31]. According to Fowler, one main idea of MVC was the concept of "Separated Presentation" [15][32]. Hereby, an application layer is separated from the GUI layer, which regularly accesses the former but not vice versa. In other words, the GUI part of a system strictly represents an independently developed sub-system, comprised of *View* and *Controller* elements, that calls the application or domain layer services by using a dedicated interface element provided by the *Model* of the MVC triad. Thus, the communication with the application layer is mostly initiated and controlled by the GUI part of a system. However, the application layer does call the GUI layer in a clearly defined way: by applying the observer pattern [18][17] *Views* are promptly updated whenever changes to the application layer or *Model* part are committed. This design allows for multiple *Views* sharing a certain *Model* and displaying different data in different ways.

In Figure 1, we present a possible architecture application diagram of the classic MVC pattern. Please note that an interface notation was used to describe the visibility (a certain set of operations) each involved design unit has on its interaction partners.

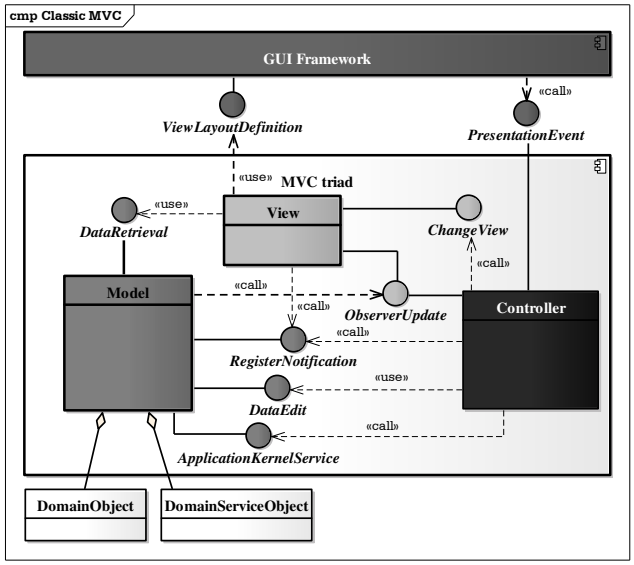


Figure 1. The classic MVC architecture pattern described by the three roles *Model*, *View* and *Controller* and their typical interfaces.

The initial setup of the triad is supported by the interfaces *ViewLayoutDefinition* (creation of screen layout, definition of UI-Controls) and *RegisterNotification*, which enables both *Views* and *Controllers* to receive notifications whenever *Model* data has changed. So, the latter is part of the observer pattern implementation. It should be considered that in the original MVC design applied in Smalltalk environments the access to the *Model* was strictly differentiated among *View* and *Controller*: the read-only *DataRetrieval* interface is used by the *View* to update its UI-Controls with current data whenever changes to *Model* have been applied. The retrieval of data by the *View* is typically preceded by a call from the *Model* via *ObserverUpdate*. In contrast, the *DataEdit* is a write-operation interface exclusively called by the *Controller* to apply changes to the *Model*, e.g., when the user has entered new data during interaction with the *View's* UI-Controls. Typical results that follow a user interaction scenario from the *Controller's* perspective are the previously mentioned change of *Model's* data (*DataEdit*), a request to the *View* to alter its display (*ChangeView*), and finally, a value creating call to the *Model* (*ApplicationKernelService*) that processes application data and changes the system's state concerning business data.

Besides these elementary interfaces and basic interaction mechanisms, the MVC pattern is affected by several problems.

Firstly, there exist many sources of the MVC pattern, which either do not cover the pattern with its multiple facets in entirety or are more or less influenced by the specific requirements of an application environment like certain GUI frameworks for either desktop or web clients. We consulted references [15], [31], [32], [33], [34] and [35] for related work. In addition, a widely accepted and often cited description can be found in [18], which was considered here of course. As mentioned in the introduction, there is no consensus on GUI patterns and their details. Ultimately, MVC ends up as "the most misquoted" pattern [15].

Secondly, the classic MVC pattern is bound to the Smalltalk environment and its basic facilities like abstract classes to create each specific member of the triad by using inheritance. As a result, the complete application was woven in MVC as a principal architecture or architectural style. This is often not applicable for nowadays system layers and current GUI frameworks. The classic MVC is conflicting and must be adapted to modern needs. For instance, Karagkasidis [10] discussed some implementation variants for a Java based MVC design.

Thirdly, from a practical point of view the classic MVC pattern misses many details that are essential to enable its benefits of modularity and separated concerns. Karagkasidis [10] already provided an elaborate examination of different concerns among popular GUI architecture patterns including MVC. In sum, the creation and assembly of GUI layout, user event handling, dialog control and the integration with business logic were identified as topics with several implementation issues.

In this regard, the MVC pattern leaves the task to decouple the three abstractions to be solved by the developer. It is noteworthy that the *Controller* is in charge of many responsibilities at once: a *Controller* has to handle the technical events (*PresentationEvent*), update the data of the *Model* (*DataEdit*), delegate the *View* to adapt its layout (*ChangeView*) to current data state, and finally, initiate the concluding processing of *Model* data by the application kernel (*ApplicationKernelService*). Therefore, this design unit is closely coupled to the *View*, as well as to the *Model*. As far as the *View* is concerned, the structure of the *Model* has to be known to enable the update of defined UI-Controls via *DataRetrieval*.

To cope with the close coupling and missing details, several variations of the MVC have been discussed [4][10]. In general, the variations in design differ concerning the distribution of responsibilities among the three abstractions. Several more patterns [6][7][18][32] occurred that mainly altered the control or introduced new concerns and abstractions. Nevertheless, they fulfill the same purpose of guiding the identification and modularization of classes in object-oriented GUI architectures. Their effectiveness can hardly be evaluated for the long term maintenance or a standardization perspective, since there are no elaborate or comprehensive descriptions available; some MVC derivates are only sourced from websites or weblogs [9][36]; comprehensive accounts on MVC variations are still under construction [37] or do not cover all variations.

B. Graphical User Interface Event Processing Chain

To be able to discuss the GUI responsibilities with increasing detail, we would like to refer to the conceptual model of the event processing within GUI architectures as described by Siedersleben [38]. In Figure 2, a variation of this model is displayed. Thereby, technical events will be emitted from the *Operating System* or later the *GUI Framework* when the user has interacted with a certain GUI element. Within the architecture, the event is either processed or forwarded by the individual components depicted in Figure 2 and the associations between components therein.

It is notable that there is a distinction of events inside the *Dialog* component. For reasons of separation of concerns, and ultimately, better maintenance of systems, the *Presentation* was assigned responsibilities with a closer connection to the technical aspects of the *GUI Framework*. Accordingly, the *Presentation* is in charge of governing the layout of the current *Dialog* and applies changes in layout, e.g., mark the UI-Controls where entered data failed the validation or activate panels when current data state requires for additional inputs. In contrast, the *DialogKernel* is to be kept independent from any technical issues as far as this is possible. So, the latter is assigned the task to communicate with the *ApplicationKernel* and its components instead.

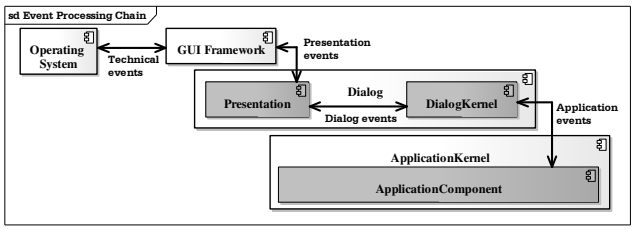


Figure 2. Value creation chain of graphical user interfaces derived from [38].

By flowing all the way from the *Operating System* towards the *Application Component*, a tiny technical event may result in the initiation of greater operations inside the *DialogKernel* or even *ApplicationComponent*. Thereby, the *Dialogs* fulfill the purpose to connect the users with the main business services provided by *ApplicationComponents*. First of all, several user inputs that result in events need to be enhanced with further information. Then they can finally be forwarded through the components to trigger business services, which create business value. That is why Siedersleben speaks of a "value creation chain" [16][38].

C. Standard Architecture for Business Information Systems

Siedersleben and Denert tended to the issues of close coupling and a better separation of concerns for GUI architectures in [29]. The main goal of their attempts was to improve the general quality of the software architecture of business information systems. With respect to the GUI, they made suggestions [29] that would prepare the standardization of the architecture of the particular domain.

Quasar. Siedersleben pushed towards further standardization attempts concerning the GUI architecture of business information systems. His efforts culminated in the creation of the quality software architecture (Quasar) [16]. Acclaimed design principles and architectural patterns, as well as the vast usage of interfaces for decoupling in combination with a new instrument for component identification were incorporated into a single software architecture manifest, which was intended to become the domain's standard.

Parts of a reference architecture [2] and the object-relational mapper Quasar Persistence [39] have been published. Conversely, the main ideas of standardization were neglected in [2] and reference architecture elements should fill the gap.

Software categories. As far as the component identification is concerned, so called software categories [16] were introduced. They consist of the five categories *0*, *A*, *T*, *R* and *AT*.

0 software designates elements that are reusable in any domain like this is applicable for very basic data types a programming language would offer.

A software is dedicated to implement a certain domain's requirements, meaning particular functions like the calculation of target costing or the scheduling of production plans for a certain machinery. So, *A* software would represent the core of each business information system.

In contrast, *T* software is responsible for the integration of technical aspects like data bases and GUI frameworks.

R software is needed whenever a technical data representation has to be converted for processing with *A* software types, e.g., a GUI string type describing a book attribute is converted to a ISSN or ISBN. In fact, *R* software also is *AT* software per definition as both domain specific and technical knowledge or types are mixed up. Thus, *AT* software should be avoided and would be an indicator for the quality of the implementation or architecture [16]. Only the *R* software used for type conversions would be permitted.

GUI reference architecture. Concerning the reference architecture portions of Quasar, the GUI client architecture [2][16] has to be mentioned for the scope of our work. Compared to common GUI architecture patterns, the Quasar GUI client architecture resembles a comprehensive architecture addressing the specific needs of a domain by incorporation of pattern elements and certain refinements.

The main parts of that architecture are illustrated by Figure 3 that is derived from [16], since this is the most detailed source available. The interface names in brackets quote the original but not very descriptive designations. The unique elements of the Quasar client architecture are the following three aspects:

Firstly, there was made a distinction of presentation and application related handling of events; the basic concept of the "value creation chain" introduced in Section II.B was developed further. Thus, there are the two design units *Presentation* and *DialogKernel* that resume original MVC *Controller* tasks besides other ones. The software categories mark both units according to their general responsibilities: the *Presentation* possesses the knowledge how certain data is to be displayed and how the user may trigger events. In contrast, the *DialogKernel* determines what data needs to be displayed and how the application logic should react to the triggered events. The communication between them is exclusively conducted via three *A* type interfaces.

Secondly, the Quasar client introduces relatively detailed interfaces and communication facilities between components compared to other GUI patterns.

To be able to fulfill its objectives, the *Presentation* relies on the *ViewDefinition* interface to construct the visual part of the dialog. Via *InputDataQuery*, the current data stored in the technical data model of respective UI-Control instances can be altered or read by the *Presentation*. Events emitted from UI-Control instances are forwarded to the *Presentation* with the operations of *PresentationEvent*.

The interfaces between *Presentation* and *DialogKernel* are mainly concerned with event forwarding and the synchronization of data between both components.

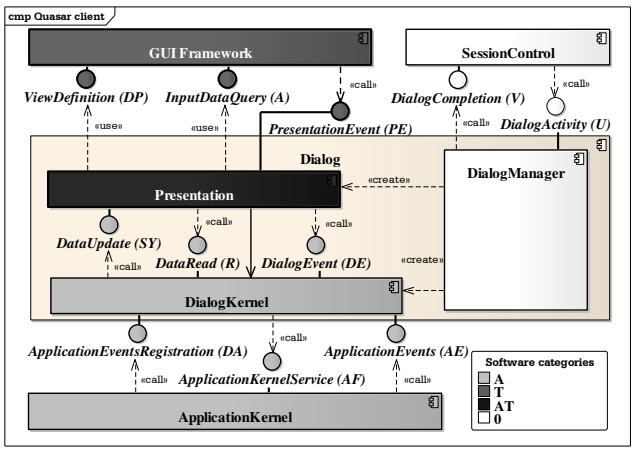


Figure 3. The Quasar client architecture based on [16].

In detail, *DialogEvent* is called by the *Presentation* whenever the *DialogKernel* has to be notified of an event relevant for application logic processing, e.g., a command button like OK or a search for available data was initiated. The Quasar client foresees two options for data synchronization. This communication step is essential, since both components possess different knowledge, and thus, work with different data structures, what is marked by the different software categories. Either the *Presentation* could read current data via *DataRead* or the *DialogKernel* would update the *Presentation* by the means of *DataUpdate*. This design shall decouple the application logic from technical aspects found inside *Presentation* and its interfaces for interaction with the current *GUI Framework*.

Thirdly, aspects that are concerned with surrounding components are also described with the Quasar client. These are interfaces dealing with the construction, deletion of dialog instances (*DialogActivity*) and reporting of results (*DialogCompletion*). Furthermore, a *DialogKernel* can register for notification (*ApplicationEventsRegistration*) about events (*ApplicationEvents*) originated from *ApplicationKernel*. To create value relevant for business logic, the interface *ApplicationKernelService* is called by the *DialogKernel*. There are more interfaces available for the coordination of transactions and the checking of permissions via an authorization component. For more details, interface specifications and a dynamic view on the architecture, please consult [2].

III. General GUI Responsibilities Model

A. Problem Statement

As we learned from the introduction, standardized GUI architectures are not available, so that custom architectures prevail. Accessible architecture sources remain only as references to be adapted to specific requirements besides standardization efforts. The basic GUI patterns and the more detailed Quasar client reference architecture are too abstract and general to describe detailed responsibilities required for implementation purposes. Hence, our conclusion from Section II is that developers have to select and always adapt a MVC or other GUI architecture pattern variant suitable for their domain.

Although the available sources will not model an extensive set of GUI responsibilities, they provide a basic set of tasks and associated components. A closer examination of given sources proved that they may complement each other, as some sources are more focused on certain responsibilities than others. A common intersection of responsibilities can easily be found. However, it is challenging to enhance this intersection in order to obtain an almost complete set of GUI responsibilities.

Finally, those GUI responsibilities have to be modeled in a systematic way, but independently from any specific pattern or framework. The target architecture for UIPs has yet to be created and it would be of little use to modify existing architectures without having identified the prevailing design issues. In addition, the influence of UIPs on the target architecture and these issues can only be understood when a complete set of GUI responsibilities was identified.

The software categories of Quasar, which were introduced in Section II.C, can serve the purpose of modeling the GUI responsibilities, since they were invented to model the occurring concerns of a system's architecture prior to the identification of components. In the following section, we will review this concept.

B. Quasar Software Categories Reviewed

We found that the concept of the Quasar software categories is ambiguous. They promise to be an instrument for component identification and quick software quality assessments. Nevertheless, they were not provided along with a clearly defined method for their proper definition or application.

Relationships. The software category types defined by Quasar can be applied for the very basic valuation of architectures, since they symbolize a very rudimentary separation of concerns between neutral (*0*), domain (*A*) and technical related (*T*) as well as mixed domain and technical (*AT*) concepts. Figure 4 displays those basic categories and their relationships. The dependencies in Figure 4 symbolize, which specialized category is derived from a more basic one. In this regard, *0* software is the parent category to be used for the composition of every other category. The elementary data structures and operations of *0* software are used to form other and often more complex data structures with their specialized operations that are unique in their purpose, which designates their final categorization.

Refinement. The further and project relevant refinement of the basic categories *A* and *T* will eventually lead to a much more powerful representation of design criteria like cohesion and coupling or design principles like modularization as well as hierarchy. During refinement each category will symbolize a certain concern of system. In this regard, "concerns" represent heavily abstracted requirements and related functions. Siedersleben [16] states that each software category ideally acts as a representative for a certain delimited topic. Consequently, the preparation of components with the aid of software category trees shall help to create high cohesive and encapsulated design units.

Complexity. By refining parent categories, a number of child categories are created that directly depend on each parent category and implicitly take over the dependencies of their parents. Following that way, it is ensured that every category may access the basic programming language facilities modeled by the *0* software category. Moreover, Siedersleben [16] speaks of complexity when refinements are created. It is obvious that refined categories truly create more complex units of design, since they potentially contain or access their own knowledge with the addition of all ancestor parent categories.

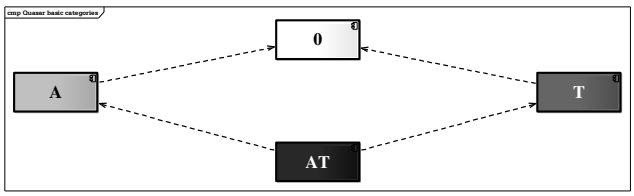


Figure 4. The basic software categories of Quasar [16].

Traceability. On that basis, refined software categories can be used to judge the purity of traceability-link [19][20] targets, meaning that these artifacts will be examined with respect to their responsibilities. When a target is made up of a mixed category, in the worst case *AT*, then it will be considered either as a model lacking detail or a design that is harder to maintain, since the developers will eventually separate the concerns during implementation by themselves. The latter is a major aspect besides the identification of potential components; that is why we consider software categories as a relevant marker.

In sum, software categories can be useful to reduce the complexity while tracing requirements to design: the categories could be kept in order to mark certain design elements inside traceability-metamodels, which are outlined in [20]. Thus, the general or refined responsibilities of design elements will be visible, so traceability-link targets can be more detailed and better differentiated.

Problems. A major problem lies in the definition and segregation of software categories. It was not clearly defined what elements drive the creation and delimitation of a software category. According to known sources [16][21], this might either be specialized knowledge how to handle certain algorithms and data structures or dependencies of an entity.

Moreover, there are only few examples [16] that explain the proper usage of the software categories. The related sources about Quasar [2][38][40][41] only use the basic software categories to mark components, but do not establish a category tree with refinements like this is done for a card game in [16].

Lastly, there exists no standard modeling concept for the software categories of Quasar. This issue could be regarded as a problem in analogy how to model architectures or identify classes. One could imply that the software categories miss comparative hierarchical concepts for their modeling like they are available for common design of architectures: architectural styles drive the identification of components. The inner design of greater components can be guided by selected architectural patterns (MVC can be given as an example). Consequently, the patterns with their defined roles drive the identification of classes and the latter serve to instantiate needed objects. However, nothing comparative is mentioned by available sources about the software categories of Quasar.

In sum, missing aspects for software category modeling are the following:

- Software category definition and delimitation,
- Software category identification approach,
- Software category standard modeling levels or style, arrangement for ease of readability or understanding

In the next sub-section, we will try to resolve these issues of software category modeling as far as our gathered knowledge and experience on this concept will guide us.

C. Rationale on Software Category Modeling

In this section, we will have to cope with the previously described problems of the software category modeling. We will have to find a way how a fine-grained responsibilities model based on the software category instrument suggested by Quasar can be established.

1) Software Category Modeling Purpose

The software categories are intended to refine tasks and document gaps left open by the available patterns. According to the Quasar rules and ideals [16], the category model to be created will represent a model with least coupling and cohesive elements that allows for planning dependencies among potential units of design. The categorization will be used in analogy to the suggested identification of components [16]; this step is essential to maintain separated concerns between identified responsibilities. Thus, the found responsibilities can be re-allocated during the development of a target architecture for UIPs or for solving common GUI design issues, but their separation can be maintained for a gain in software architecture quality and interface design with least coupling.

2) Software Category Modeling Levels

Quasar examples. With the given explanations, the software categories' scope remains vague. Therefore, we analyzed the provided example software category trees in reference [16]: on the one hand, some trees model abstract concepts like GUI, Swing and data access. On the other hand, the categories are applied to express certain component instances of a particular sub-system, as this is shown for an application kernel component dedicated to services a book library would offer.

From these examples, we conclude that a category tree can be situated on two principal levels of refinement: a software category tree that models abstract concepts and a tree, which is used to represent certain instances of a chosen concept of the former, are to be differentiated.

Abstract concept tree. The abstract description level is used to identify the general areas of knowledge that occur in a system and its components. This category tree is an abstract view on responsibilities that we understand as the arrangement of meta-types, which are permitted to occur in a system. So, the software categories on that level determine what type of tasks or sets of responsibilities are to be considered. Each set of responsibilities will correspond to a certain component stereotype. We understand that level of modeling in analogy to the object-oriented (OO) class concept: software categories model meta-types for design units to be identified. As OO classes determine what kind of objects can be instantiated, the software categories establish the types of design units, which define the software architecture's structural components.

The software categories of the abstract level are derived from the two basic categories *A* and *T*, and thus, the fundamental areas of knowledge of domain specific logic and technical interactions within the software architecture. Figure 5 illustrates an example for an abstract software category tree and its meta-types. Each meta-type expresses a set of tasks or responsibilities like this is the case for categories like *GUI dialog component*, *Application kernel component* and *File system persistence*, which express that layers or even components fulfilling the general task of proving application logic, a graphical user interface and file

system based access will be present in the system. This kind of modeling of software categories can be understood as principal or general architecture modeling where the required layers and major component types are identified. In other words, the abstract software category tree answers the following question: what layer, component or other types of design units do occur in a system?

The sub-categories of the meta-types will be the actual layers, components, classes, and operations depending on the chosen detail in the hierarchy of modeling. According to traceability concepts mentioned in the previous section, the meta-types symbolize traceability-link targets in a taceability-metamodel: these are the principally allowed target types. For instance, a primary and simple distinction based on Figure 5 can be made between application and GUI components. So, requirements can either be associated to one of each type. This distinction is rather simple, but more effective than just allow the requirements to be traced to any arbitrary design entity. Another example can be derived from Evans' [22] domain model stereotypes. He identified concepts like services, entities, factories and value objects. These are abstract, but more concrete than arbitrary design units, and could be modeled in a software category tree as meta-types. Any other pattern type that has distinctive roles and their tasks described could be modeled with an abstract software category tree. In this regard, patterns with their set of characteristic classes can fill the gap that exists between components and bare classes: with the aid of software categories they permit the modeling of collaborations.

The sole modeling of one pattern makes little sense as the pattern's own description would suffice and most likely would be more detailed rather than a corresponding software categories tree. However, the modeling of system specific meta-types and the integration of patterns could be beneficial. Thereby, the categories would express the sum of all potential instances and the fact, that a certain pattern is present at a certain level in the systems's hierarchy of needed or allowed software categories. In addition, the software categories could be used to arrange a certain pattern and its roles in order with the existing hierarchy of design units. As result, the single roles or elements of a pattern do not need to be allocated to a fixed design hierarchy level like OO classes; they could be assigned to components as well.

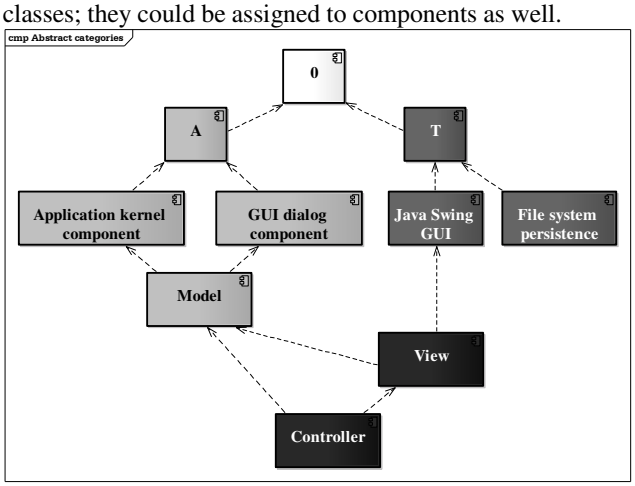


Figure 5. An example of an abstract software category tree.

This approach could be used for the refinement or even the combination of several patterns to structure a custom hierarchy or collaboration of classes.

For the sake of the example, Figure 5 was detailed with the categories *Model*, *View* and *Controller* to express that the MVC pattern (see Section II.A) will be applied in this system. In addition, the influences for that specific pattern application were added as dependencies among the software categories.

Accordingly, the *View* will be determined both by knowledge how to build visual forms with *Java Swing GUI* and how to properly access (assignment of data to UI-Controls in the correct order) the business data provided by the *Model*. In addition, *View* is implicitly determined by knowledge about the system's GUI specification with required layouts for certain functions or use cases represented by the *GUI dialog component* category. By maintaining the dependency to the *Model*, the *View* implicitly is connected to the parent categories including *GUI dialog component* on higher levels up to the basic *0* category, which is needed for the realization of every software category. Moreover, the *Controller* category is both influenced by the *Model* and *View* category: to perform data changes and coordinate application service calls, the dependency or knowledge of the *Controller* category must span the *Model* internals. The dependency on the *View* expresses that the *Controller* has to know about the *View's* structure or state to be able to request a proper change of the current screen layout or react on a certain UI-Control event trigger. The knowledge on *Java Swing GUI*, which is required for the *Controller* to be able to implement GUI framework specific event listener interfaces, is incorporated implicitly with the dependency on the *View* category.

However, this example points out what difficulties may occur by the integration of GUI or architectural patterns in a custom component architecture. Foremost, the three categories *Model*, *View* and *Controller* symbolize rather abstract concepts as they are described by the sources mentioned in Section II.A. More details about these three stereotypes have to be revealed in order to prepare the derivation of system specific instances and their implementation. Therein the difficulties are situated, whilst there is no consensus about the further refinement of each category or pattern role. Since acclaimed sources [3][18][32] do not provide sufficient details for current requirements, several different refinements [4][10] or interpretations for the MVC exist that result in varying dependencies and may differ from our example in Figure 5. Thus, the inner structure of each MVC category is not clearly determined and may vary as well. So will be the final quality of architecture and the separation of concerns depended on individual refinements. We could further detail each MVC category to achieve a clear distribution of responsibilities and guide the identification of smaller design units such as interfaces, classes and operations. This step can be quite helpful, since components are the ordinary corresponding unit of design for software categories [16], but these units are to be assigned to available programming language elements. Common programming languages do not feature a component as a unit for implementation after all.

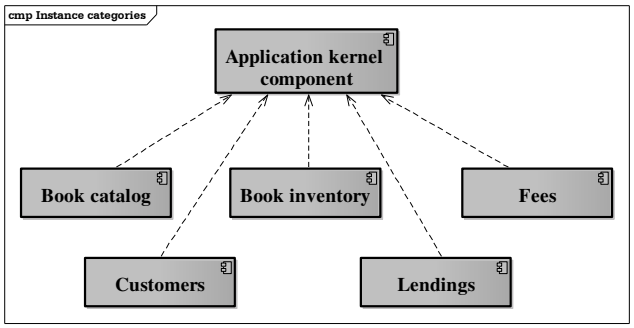


Figure 6. An example software category tree derived from [16] displaying identified components on the basis of a meta-type software category.

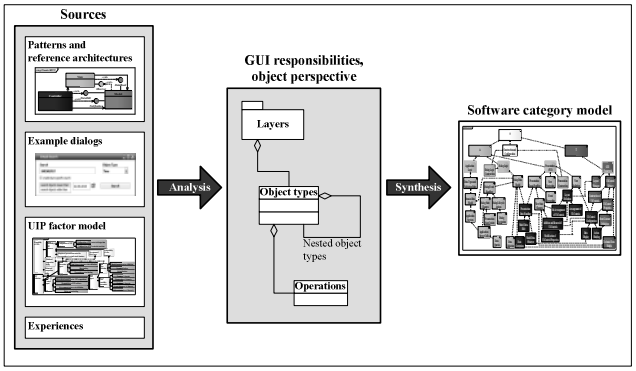


Figure 7. Process applied for the derivation of GUI software categories.

Refined software categories on the basis of certain class collaborations provide a modeling level in between and may fill the gap.

Instance tree. A second modeling level of software categories can be applied on the basis of the abstract concept tree. When the meta-types have been identified, the system specific instances or actual components or even classes need to be identified based on the found categories.

For instance, a software project would need 20 dialogs to appear in a system, which would contain 30 *View* instances, since 10 dialogs each would require two separate *Views*. These categories with their scope set to instances resemble concrete traceability-link targets in a project that are part of certain associations or dependencies.

Figure 6 displays an example based on Figure 5 where the abstract meta-type category *Application kernel component* was detailed with five needed instances as sub-categories. One could insert a suitable pattern for *Application kernel component* in Figure 5 like this was done for the MVC. Maybe Evans' domain concepts [22] could detail the *Application kernel component* as mentioned above, but this would alter the level of detail of Figure 6 as well.

It is obvious that the relationships of Figure 6 are rather simple and stereotype. We are inclined that the instance categories may introduce relationships among each other and eventually alter the dependencies inherited from the abstract parent software category. But these considerations are out of the scope of this work.

Summary. We outlined how the software categories of Quasar can be used to describe patterns in more detail or independently describe their responsibilities. We will tend to the described pattern refinement problem, and so, follow a similar way as seen in Figure 5. Our idea is to compose the GUI responsibilities from several sources at once and make use of an abstract software category tree to arrange them in an appropriate way. So, the categories will serve as the means for structuring, grouping and proper separation of responsibilities.

3) Software Category Identification Approach

We seek to establish a basis for the responsibilities that are regularly discovered in a GUI architecture. Our approach is depicted in Figure 7.

In detail, we will rely on four different kinds of sources and analyze them to identify the GUI responsibilities:

relevant responsibilities will be derived mainly from related work about patterns and reference architectures; considered sources are references [3], [4], [5], [6], [7], [8], [10], [14], [15], [16], [18], [29], [31], [32], [33], [34], [35], [36], [37] and [38]. In addition, we rely on more sources not described in this work: we use sample dialogs, consult the UIP factor model [42] and integrate own experiences from software development projects.

In fact, we do a decomposition of GUI architectures to rather atomic object types, related operations and nested object types. These entities will be separated and delimited in order to establish a unique software category tree. We examine, what can be solved with *0* or *A* software and what concerns are definitely dependent on GUI framework code (*T* software). Finally, the found responsibilities will be assigned to individual software categories, which will be used according to the rules of Quasar to synthesize an abstract software category tree displaying GUI architecture responsibility concepts.

Basic software categories. As the software categories are not clearly defined in original sources, we will have to point out how to create new and delimit existing software categories.

On the root level, we will comply with Quasar and use the basic categories *0* (white), *A* (light grey), *T* (medium grey with white caption) and *AT* (dark grey with white caption). The basic category *Construction and Configuration* was added to represent the creation of new objects as well as the configuration of interfaces with implementing objects.

On the next level, layers and technological boundaries of the application architecture are represented. With that step, the main ordering concept of the analysis in the middle column of Figure 7 is established. Finally, the main layer categories *Application Kernel*, *Dialog Logic*, *Presentation* were identified as *A* category children, since they depend on the individual domain-specific requirements of a software system. Moreover, *Presentation* and *Dialog Logic* were separated as software categories according to the event processing chain of Figure 2.

Category identification. As the tree gets more detailed, software categories will become very fine grained and embody components, collaborations, classes or even operations. Since the categories can distinguish components and their dependencies, they could be applicable for the delimitation of the smaller units of design, too.

To identify each of the refined categories, we applied several rules of thumb. During the analysis of GUI architectures and patterns, we derived categories from the different families of operations that regularly occur in the scope of certain units of design. In general, these were the definition or modification of object types or their data types, event triggering or processing, as well as forwarding of both data and events. These kinds of operations occur for different layers like technical or application related objects of general GUI pattern components that are common for MVC or the Quasar client. The different layers symbolize certain levels in the software category tree and were derived from reasonable abstractions like application logic, presentation logic and presentation technology. These layers should help us to prepare a coarse-grained order principle of GUI responsibilities and let us establish a high level categorization. The applied layers are partially related to the ones outlined in [10]. We alter and extend the given description. The layers will be explained in the following listing:

- *Application logic*: The objects and operations are dedicated to realize the core functional requirements of a business information system.
- *Presentation logic*: This layer is settled in-between the two other layers. So, it resumes tasks that cannot clearly be assigned to one of the other layers. These tasks include the handling of states that affect the visual appearance and navigation among different screens or dialogs. Furthermore, the logic that determines what application logic calls are appropriate in a certain state or how data states influence the screen layout and its UI-Control states is realizes in that layer. In sum, it couples application logic and presentation technology layers to create a seamless flow of interaction. This is done by translation of events emitted from presentation technology to application logic services and data changes. Changed data has to be reflected on the presentation technology layer; hence the presentation logic has to initiate a respective update of the presentation technology layer. Basically, this layer addresses the need that the different components on the various layers do influence each other with their internal state changes as this is described by Karagkasidis [10].
- *Presentation technology*: Both GUI framework and system objects are combined to create or alter the views and visual effects of a GUI for displaying data and interaction facilities. The visual states are implemented with that layer's objects and operations, but its tasks do not include the logic required for deciding what state is appropriate in a certain situation. In addition, the reaction to user inputs and the activation of event processing are further tasks of that layer.

For each of the layers, we distinguished the belonging operations and data structures according to the knowledge and types required for their processing. When operations demanded for the usage of certain types in a context that was not in scope of the originator then categories were definitely of a mixed kind. In contrast, categories were left pure when interfaces using neutral *0* or *A* types could be used for delegations. A hint close to implementation considers what would be the import declarations in a unit of design with respect to Java language. If the import was based on interface types using neutral *0* types, the category would remain pure. The software category would be mixed, if the imports will demand for the addition of types defined exclusively in the imported unit of design.

These considerations led us to finding software categories on different levels of an abstract software category tree; it also inspired us to establish a clear definition of software categories that is presented in the following sub-section.

4) *Software Category Definition and Delimitation*

So far, the fundamentals surrounding software categories were described. It is still to be declared what are the concrete contents of a software category. This aspect is essential to describe each software category's individual details and to be able to delimit them.

However, the clarification of software category contents is not supported by available sources. Therefore, we derived certain dimensions that exist in a hierarchical order of dependency. These dimensions outline the contents of one specific software category. Figure 8 illustrates what dimensions define the knowledge that resides inside software categories. The following paragraphs will explain each of the dimensions.

Specific content. Each software category is dedicated to a specific topic or area of knowledge. All sub-ordinate dimensions depend on the choice of that content. Thus, the dimension acts as a filter to permit the inclusion or exclusion of certain *Contained entities,* what *Type of operation* is to be performed with them, and finally, what *Knowledge* must be possessed for the implementation or usage of defined operations.

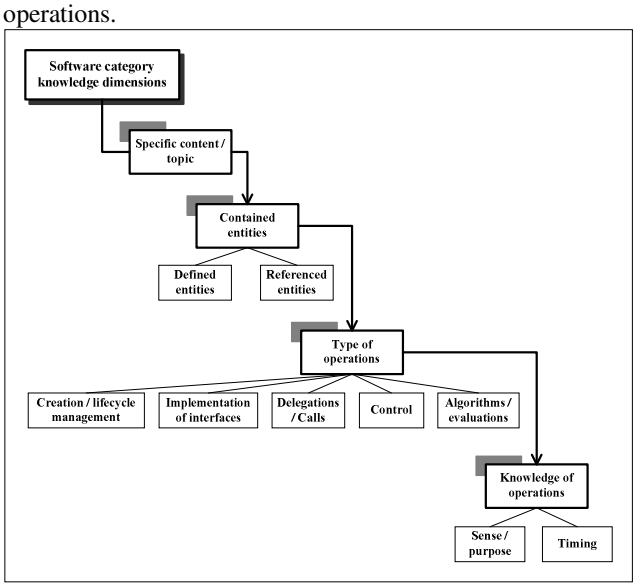


Figure 8. Software category definition via successive dimensions.

For instance, the software category *Java Swing GUI* of Figure 5 permits the containment of every class of the Java Swing GUI framework and all basic Java foundation classes that can be assigned to *0* Software further upwards in the tree.

The expression of that dimension can hardly be formal. A semi-formal approach may be established by assigning certain requirement models as the specific content.

Contained entities. This dimension determines what object types or units of design are to be considered inside the software category. Two particular cases are to be differentiated: in the first case, a software category may introduce and define specific units of design. These originate from and exist in the scope of that particular software category. In the other case, the software category is referencing entities or units of design that originate from other software categories and are not defined inside the current software category. This case often occurs for the import of interfaces connecting different components or classes or for the incorporation of foreign data structures.

The entities may consist of layers, components, interfaces, classes or even smaller units of design. It largely depends on the hierarchy level the software category resides on. To constrain the set of entities the first dimension puts up a global limitation for the software category. The scope of topics, and so, the number of contained entities differs greatly with respect to the given software category hierarchy level. Thus, the concept of software categories follows a hierarchical de-composition downwards the tree. It is of the essence for each architect to find a suitable level for detailed modeling to achieve proper cohesion and no coarse or too fine-grained units of design.

Besides, the second dimension affects the third dimension in a way that objects and data structures both for calling (allowed parameters) and implementing (interfaces and data structures) operations are defined. According to the refinement of software categories, the dependencies of the current software category express that all *Contained entities* from the previously defined parent categories are implicitly contained as *Referenced entities*.

With the given definition of the second dimension a software category may formally be defined by the entities it contains or references.

Type of operations. The next dimension is concerned about the general type of operations performed with the previously *Contained entities*. There are various options:

- *Creation*: Entities are created with the knowledge of the software category. Additionally, the entire lifecycle of entities may be governed.
- *Implementation*: Interfaces required to interact with certain entities are implemented. These can be call-back interfaces that are typical for the event listeners of GUI frameworks. Furthermore, interfaces can be defined by superior entities that need a certain set of operations to be implemented by lower situated interaction partners.
- *Calls / delegations*: Operations of other entities are called and the control is passed on to them.
- *Control*: Other entities are called with their operations but the control remains inside the software category. This kind of operation is applied in order to coordinate a flow of operations or events.
- *Algorithms*: Domain specific calculations are performed or technical routines activated. The results are obtained from the knowledge present in the software category or are gathered from *Referenced entities* operations that may eventually be used for enhancement or aggregation.

Depending on the type of operation combined with the considered entities, the software category type, its purity or coloring may change. For instance, the *Controller* of Figure 5 is no pure category, but of a mixed type, since it controls both the appearance of the *View* (compare *ChangeView* interface of Figure 1) and simultaneously coordinates calls to the Application kernel components (compare *ApplicationKernelService* interface of Figure 1). So it must possess knowledge about both topics at once. In addition, the *Controller* has to implement event call-back interfaces that are referenced within its scope but are defined in and constitute parts of other software categories like *Java Swing GUI*.

Both the second and third dimension can be sharply determined and delimited by enumeration of entities and operation types performed with them. Therefore, the two dimensions together represent the formal part of a software category's definition.

Knowlegde of operations. This final dimension expresses the proper moment in time and purpose of a contained operation inside the software category. Essentially, it represents the proper sequence, atomic steps and meaning of operations. This knowledge combined with the definitions of the previous dimensions embodies the ability to finally implement the operations of a software category.

D. Graphical User Interface Software Category Model

In this section, we will apply the approach presented with Figure 7 before we describe the GUI software category tree. Hence, the following sub-sections will analyze the responsibilities covered by GUI architectural patterns and their sources introduced in Section II. We will begin with the MVC and its variants, which is followed by the analysis of the Quasar client.

1) Analysis of the Model View Controller Responsibilities

The responsibilities described by the MVC pattern and its variants are summarized in TABLE I. Please note that the sources for the different MVC responsibilities are not completely mentioned; only the primary or sources with significant descriptions of these responsibilities are considered. Moreover, the assignments of operations may vary due to several MVC design options, which are exemplarily described in [4] and [10]. Our scope is the completeness of responsibilities and not the display of different design options.

2) Analysis of the Quasar Client Reference Architecture Responsibilities

Compared to the previously illustrated MVC responsibilities, the Quasar client includes many of these but considerably adds detail concerning the presentation logic and application logic layers. According to Siedersleben [16], the Quasar client compares to MVC as follows: the *View* is contained in the *Presentation*.

TABLE I. MODEL VIEW CONTROLLER PATTERN RESPONSIBILITIES.

Pattern role	Responsibility	Operations	Defined / referenced entities	Layer
Model	stores business data [18][31], provides results of data queries or intermediary object data [31]	read model data, change model data	Defined / referenced: data read and write interfaces for business objects and data types, aggregates or selections of business objects and their attributes (intermediaries [31]) (Inclusion or references depend on the realization of the model as a part of the application / business layer or as a separate unit of design.)	application logic
	validate data [4][35], provide additional information for visual interpretation of data [15]	validate data, read data interpretation information	Defined: data interpretation information Referenced: business data types and validation information	application logic
	provide an interface for calling application services [18]	call application service	Referenced: application services, business objects and data types as parameters	application logic
	register observers to be notified upon data changes [18]	register observer, deregister observer	Defined: list of observers Referenced: observer interface	presentation logic
	notify observers about data changes [18][31]	notify observers	Defined: setChanged interface Referenced: observer interface	presentation logic
View	display data, information and functions [18][31], arrange screen layout [31][15], visually interpret data [15], highlight validation errors	display initial screen, change screen layout, read model data, interpret model data	Defined: possibly specializations of GUI framework classes (may be used for data interpretation) Referenced: UI-Controls and layout managing facilities provided by the GUI framework, model data	presentation technology
	update data display [18]	read model data, update UI-Controls	Defined: update display interface Referenced: UI-Controls provided by the GUI framework, model data	presentation technology
	transform business data to technical GUI data model [31][35][16]	read model data, transform data	Referenced: model data, UI-Control data models required by the GUI framework	application logic, presentation technology
	create corresponding controller [18]	create controller	Referenced: associated controller	presentation logic
	register as observer of the model [18]	register observer, deregister observer	Referenced: model observer interface	presentation logic
	composition of hierarchical views [18][10]	create sub-view	Referenced: subordinate views, UI-Controls provided by the GUI framework	presentation technology
Controller	receive and react to user input [18][31]	handle event	Referenced: event listener interface provided by the GUI framework, possibly view's UI-Controls to determine event source and react differentiated	presentation technology
	translate events to service requests of either model or view [18][31]	call model service, change model data, call view display update	Referenced: model service interface, model data interface, view state change interface	presentation logic, presentation technology
	register as observer of the model [18]	register observer, deregister observer	Referenced: model observer interface	presentation logic
	update upon receiving notification from model [18]	update controller state, update view state	Defined: update controller interface Referenced: view state change interface, model data	presentation technology, presentation logic

TABLE II. QUASAR CLIENT REFERENCE ARCHITECTURE RESPONSIBILITIES.

Pattern role	Responsibility	Operations	Defined / referenced entities	Layer
Presentation	display data, information and screen layout, provide a proper localization	display initial screen, change screen layout (DP), read dialog data (R)	Defined: possibly specializations of GUI framework classes (may be used for data interpretation), presentation data model Referenced: UI-Controls and layout managing facilities provided by the GUI framework, dialog data model, localization data	presentation technology
	react to user input	handle presentation event (PE)	Referenced: event listener interface provided by the GUI framework	presentation technology
	forward events to dialog kernel when events are out of presentations' scope, attach event data	forward dialog event (DE), forward event data	Referenced: dialog event interface, presentation data model	presentation logic
	update upon receiving notification from dialog kernel	update presentation state (SY)	Defined: presentation data model, update presentation state interface Referenced: UI-Controls provided by the GUI framework, dialog data model	presentation technology, presentation logic
	control presentation states and trigger changes in screen display	change presentation state	Defined: presentation states Referenced: UI-Controls and layout state, presentation data model	presentation technology, presentation logic
	transform dialog data to presentation data	read dialog data (R), transform data	Defined: presentation data model Referenced: dialog data model, UI-Control data models required by the GUI framework	application logic, presentation technology
	validate input data to ensure proper formats are entered by the user	validate presentation data	Defined: presentation data model Referenced: business data types and validation information	application logic
Dialog kernel	handle dialog events, control dialog states and dialog lifecycle	forward dialog event (DE), change dialog states, close dialog, open sub-dialog	Defined: dialog states model Referenced: sub-dialogs	presentation logic
	control data states and retrieve data from the application kernel	ApplicationKernelService (AF), update dialog data model, update dialog state	Defined: dialog data model Referenced: application data model, application data queries	presentation logic, application logic
	notify presentation about data changes	update presentation state (SY)	Referenced: update presentation state interface	presentation logic
	translate events to service requests for the application kernel	ApplicationKernelService (AF)	Defined: dialog data model Referenced: application kernel service interface, application data model	application logic
	update upon receiving notification from application kernel	ApplicationEvents (AE), update dialog state, update dialog data model	Defined: dialog data model, dialog states model, ApplicationEvents interface	application logic, presentation logic
	register application kernel as observers of data or state changes	ApplicationEventsRegistration (DA)	Defined: ApplicationEventsRegistration interface Referenced: ApplicationKernelService interface, list of observers	application logic
	validate dialog data before calling application kernel services	validate dialog data	Defined: dialog data model Referenced: application data model, business data types and validation information	application logic
Dialog manager	control the lifecycle of the dialog composition	create and close dialog kernel, create and close presentation	Referenced: associated dialog kernel and presentation	presentation logic

Controller tasks are shared among *Presentation* and *DialogKernel*; they implement different control facilities with respect to their individual scopes (presentation technology and presentation logic). Lastly, the *Model* is realized by the data models of *Presentation* and *DialogKernel*.

In TABLE II, the responsibilities of the Quasar client, which we could reveal from references [2][14][16], are presented. Please note that Siedersleben mentions several design options in reference [16] that affect the communication between *Presentation* and *DialogKernel* (Figure 3). We based our description of the responsibilities on the architecture diagram of Figure 3; the displayed interfaces were considered in TABLE II accordingly.

3) Synthesis and Description of the Graphical User Interface Software Category Model

The resulting software category tree is depicted in Figure 10 and will be developed in the following paragraphs. It has to be considered that the software categories do model dependencies between units of design and no flow of events or algorithms. Although there will be interfaces between software categories for later implementation, these cannot be illustrated by the software category tree but can be later on determined concerning the possible type.

Principal units of GUI design. To clarify what units of design will be considered for a GUI system, we consulted the directions given by related work. Our findings were that MVC patterns often relate to single *Views* that model the visual display for a certain state of data or processing. In contrast, the Quasar client considers *Dialog* units that comprise of visual and logic components. Additionally, *Dialogs* feature an own life cycle and can activate or de-active each other, so that a flow of *Dialogs* and corresponding presentations or *Views* is established.

For a general GUI responsibilities model and its possible practical applications, the given definitions of both MVC and Quasar client were not entirely sufficient. As far as the Quasar client [16] is concerned, the relationship between input masks (or views) and dialogs is not entirely clarified, so that we received the impression that each Quasar client *Dialog* (Figure 3) is expected to have only one dialog data model and one *Presentation* (Figure 3) unit. As a consequence, we incorporated the following enhancements in the hierarchy of GUI design units:

A *Dialog* corresponds to one or more *Use Cases* of the system requirements specification and may be associated with several *follow-up dialogs* or *auxiliary dialogs* [16]. To provide data for display, processing and storing user inputs, each *Dialog* unit contains a *Dialog Data Model*. This model is closely related to the data requirements of the realized *Use Cases*. To be able to present several *Use Cases* steps individually or partition data among several views, each *Dialog* is associated with one to many *Presentation* units, which realize the corresponding display of a given *Dialog* or *Dialog Data Model* state.

From our experience, it is reasonable to keep dialog data and consecutive user interaction steps with several different displays together in one GUI design unit.

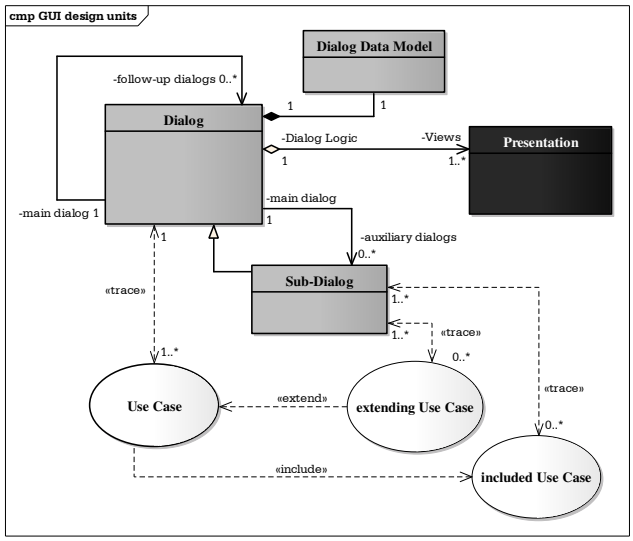


Figure 9. Principal units of GUI design and their requirement sources.

For instance, a *Dialog* may consist of a *Dialog Data Model* with several objects that cannot be displayed with one single window. Accordingly, the data is structured among several *Views*, which can be realized with different tabs of one window or with several windows. That is why each *Dialog* may reference several *Presentation* units, which serve as different *Views* (Figure 1) with their sets of UI-Controls and layout definitions. Accordingly, the user may proceed with required *Use Case* interaction steps straight forward or may return to previous steps in order to revise inputs. The data for all steps is kept together in one *Dialog* unit and its respective *Dialog Data Model*. Therefore, the communication needs between *Dialogs* concerning data exchange is reduced to a required minimum.

Furthermore, a *Dialog* may reference *Sub-Dialogs* that are closely related to either *included* or *extending Use Cases*. For instance, a search for certain objects can be added to some *Dialogs* as a *Sub-Dialog* to support the user during the selection of relevant data (*included Use Case*) for a certain context (*Use Case*). The particular search *Sub-Dialog* may appear in other *Dialogs* as well.

Figure 9 illustrates the GUI design units and their described relationships. The GUI design units were identified in correspondence to the event processing chain of Figure 2 and the basic software categories and layers of Section III.C.3) we apply for software category modeling. Thus, the *Dialog* serves the presentation logic and *Presentation* is responsible for presentation technology. Both GUI design units will lead the identification of detailed software categories and respective responsibilities within their scope of data and event handling.

The sub-trees of software categories illustrated by Figure 10 will be described with respect to their different scope as follows.

Presentation layout. The categories derived from *Presentation* are closely related to the *View* and *Controller* of the MVC pattern [18] and detail both their responsibilities. TABLE III provides a summary of the software categories modeling *Presentation* layout concerns.

Presentation is marked with FUI (final user interface) [43] given that this category symbolizes the certain knowledge required for creating the specific *View* part of a given GUI system. This category is further branched into *View Definition*, *View Navigation* and *Presentation Event Handling*. The involved software categories have to comply with project specific dialog specifications and at the same time need to possess knowledge about the types and operations the integrated *GUI Framework* offers. Hence, all sub-categories heavily depend on technical aspects. They each constitute a mixed category.

TABLE III. PRESENTATION SUB-TREE SOFTWARE CATEGORIES.

Sub-Category	Topic	Contained entities	Operations
Presentation	Visual parts of a *Dialog* that realize the presentation technology layer. defines interfaces used in child software categories for construction purposes	Defined: Presentation Construction interface, View Definition interface, Presentation Event Handling interface	Abstract
Presentation Construction	constructor of a *Presentation* unit	Referenced: Presentation Construction interface, View Definition interface, Presentation Event Handling interface	Creation: *Presentation* (*View*) units with their comprising parts of View Definition, Presentation Event Handling and View Navigation Implementation: Presentation Construction interface (activates the constructor of a *View* to enable its creation along with associated UI-Controls, layout and event handling)
View Navigation	Changes and activates the *Views* that can be part of of one *Dialog* unit. This responsibility is essential for *Dialogs* that constitute several steps with or without different choices leading to certain *Views*. *Views* shall be decoupled from each other to facilitate their exchangeability and even reuse. That is why the *View Navigation* interface has to be called from outside the *Presentation*.	Defined: View Navigation interface, states or target *Views* for navigation Referenced: Presentation interface	Creation: Creates different *Views* by calling Presentation interface Implementation: View Navigation interface (offers access for entities outside *Presentation* to trigger *View* changes)
View Definition	Visual part of a *View* that creates and holds all UI-Control and layout information	Defined: UI-Control Configuration interface, Layout Definition interface Referenced: View Definition interface	Creation: UI-Control Configuration, Layout Definition Implementation: View Definition interface (constructor)
UI-Control Configuration	construction of UI-Controls, setting of UI-Control specific properties	Defined: possibly specialized UI-Controls created by inheritance from the GUI framework, UI-Control state data Referenced: UI-Controls of the GUI framework and their properties, UI-Control Configuration interface	Creation: create UI-Control Delegation: set UI-Control property Implementation: UI-Control Configuration interface (creates the UI-Controls upon being called by *View Definition*)
Data Display	UI-Control specific display of data, interpretation of model data [15] like coloring and highlighting of validation errors The dependency to the GUI framework *Technical Data Models* is derived from *Model Data Observer* and its parent software category *Presentation Data Handling*.	Defined: UI-Control display data (for simple data display and interpretation of data) Referenced: Dialog Data Model read interface Technical data model interface	Delegation: Read Dialog Data Model Algorithm: Interpret Dialog Data Model Control: change the technical data model and associated display of UI-Controls based on Dialog Data Model contents and its interpretation
Layout Definition	Creates and defines the layout of the *View* The category itself is abstract, so that its child software categories do the actual implementation of layout creation. Thus, the child categories can be regarded as different strategies of the *Layout Definition* interface.	Referenced: Layout Definition interface, Layout managers of the GUI framework, UI-Controls of the *View*	abstract
Arrangement of UI-Controls	Creates the general *View* layout, assigns layout to containers like panels, parts, cells positions UI-Controls inside layout containers	Referenced: Layout Definition interface	Implementation: Layout Definition interface (creates the *View* layout upon being called by *View Definition*) Algorithm: create *View* layout with the help of layout manager operations

The *View Definition* category is detailed with the responsibilities required for the initial creation of the visual parts of a *Dialog* and the declaration of layout specific elements. We separated the *Layout Definition* and *UI-Control Configuration* as the layout aspects often involve the usage of dedicated objects and operations that considerable differ from the instantiation and configuration of UI-Controls. For the reasons that events require dedicated operations and not all created UI-Controls have to be bound to certain events, the category *Action Binding* was separated as a specialization of the *UI-Control Configuration*.

View Navigation enables the change of different *Presentations* of a *Dialog* with respect to Figure 9.

Data Display was added to better reflect the visual presentation of data, which was formerly [1] included in *UI-Control Configuration* (setting properties for data values), and includes the interpretation of certain data values as an additional responsibility derived from [15].

Presentation event handling. The *Presentation Event Handling* category serves the task to receive and evaluate *Presentation events* according to Figure 2. It is branched into *Presentation Data Handling*, *View State Changes* and *Event Forwarding*. The first child handles both the reading (*Model Data Observer*) and editing (*Model Data Edit*) of *Dialog* data from the *Presentation* perspective. The changes in layout, properties and arrangement of active UI-Control instances during runtime are optional tasks that are embodied by the category *View State Changes* and its children. Certain events cannot be further processed by the visual dialog units, so that they need to notify the next unit in the chain of responsibility. This rationale is based on Figure 2. The required knowledge how to react to any received events is concentrated in *Presentation Event Handling*. Its child software categories serve the above described responsibilities on demand of the superior evaluation of *Presentation Event Handling*. For instance, the decision about what respective events are to be forwarded is made by *Presentation Event Handling* and the actual forwarding command is encapsulated by *Event Forwarding*.

In TABLE IV, the software categories responsible for *Presentation* based event handling are summarized.

TABLE IV. PRESENTATION EVENT HANDLING SOFTWARE CATEGORY SUB-TREE.

Sub-Category	Topic	Contained entities	Operations
Presentation Event Handling	Event handler called by an UI-Control with active *Action Binding* This software category evaluates any incoming events from UI-Controls and decides on a proper reaction: *Presentation Data Handling*, *View State Changes* or *Event Forwarding* are triggered. For instance, it decides what events can and cannot be processed by the *Presentation* and must be forwarded to the *Dialog Event Handling*. Just the decision is covered here, how the forwarding is performed is in the scope of the respective child software category.	Defined: Event Forwarding Interface, View State Change Interface, Presentation Data Handling interface, Action Binding interface Referenced: Presentation Event Handling interface	Implementation: Presentation Event Handling interface (constructor), Action Binding interface (to be notified of any event intercepted by UI-Control Action Binding) Algorithm: Determine the proper reaction in response of the received event Control: Activate the proper reaction implemented by its child software categories
Action Binding	definition of various event listeners for UI-Controls to enable a reaction to specific events The event is just intercepted by the implementation of the event listener interface. Eventually, the resulting reaction is not covered but prepared with the delegation to the presentation event handling.	Referenced: Event listener interfaces of the GUI framework, Action Binding interface	Implementation: specific event listener interfaces of the GUI framework Delegation: call Action Binding interface to notify Presentation Event Handling about user inputs
Event Forwarding	forwards events to the *Dialog Event Handling*	Referenced: Event Forwarding interface, Dialog Event Handling interface	Implementation: Event Forwarding interface Delegation: forward event (Dialog Event Handling interface)
View State Changes	Interface that permits the change of *Presentation* states, which can be called by the *Presentation State Update*. May be called for changes like the activation of hidden or collapsed panels. The possible states a *View* can adopt are modeled by this software category.	Defined: Interfaces of child software categories (Re-Arrangement of UI-Controls, Modification of UI-Control Properties, Addition and Removal of UI-Controls), *View* state model Referenced: View State Change Interface	Implementation: View State Change Interface Control: Call appropriate child interface to enable the appropriate change of visual state
Re-Arrangement of UI-Controls	Change the position of UI-Controls inside the *View* layout on request of *View State Changes*	Referenced: Re-Arrangement of UI-Controls interface	Implementation: Re-Arrangement of UI-Controls interface Algorithm: alter *View* layout with the help of layout manager operations

Sub-Category	Topic	Contained entities	Operations
Modification of UI-Control Properties	activate, hide, or change UI-Controls in size, color or any other visual property on request of *View State Changes* May be called when data validation failed or new data state requires the update of particular UI-Controls only. In addition, UI-Controls can be set to be read-only when no further editing shall be permitted.	Referenced: Modification of UI-Control Properties interface, UI-Control state data, UI-Controls of the GUI framework and their properties	Implementation: Modification of UI-Control Properties interface Delegation: set UI-Control property
Addition and Removal of UI-Controls	change the set of active UI-Controls of a particular *View* on request of *View State Changes* UI-Controls may be added or removed as a result depending on the current data state or events evaluation.	Defined: possibly specialized UI-Controls created by inheritance from the GUI framework, UI-Control state data Referenced: Addition and Removal of UI-Controls interface, UI-Controls of the GUI framework and their properties	Implementation: Addition and Removal of UI-Controls interface Creation: create UI-Control, delete UI-Control Delegation: set UI-Control property
Presentation Data Handling	event handling that is only concerned about data changes and storage from the *Presentation* point of view	Defined: Model Data Edit interface, Model Data Observer interface Referenced: Domain Data Model, Technical Data Models of the GUI framework, Dialog Data Model interface, Dialog Data Model observer registration interface	Delegation: register as observer with *Dialog Data Model* Algorithm: determine proper data handling reaction Control: initiate data update via Model Data Observer interface, Activate Model Data Edit interface
Model Data Edit	changes *Dialog Data Model* in order to store user inputs present in active UI-Controls	Referenced: Dialog Data Model write interface, Model Data Edit interface	Implementation: Model Data Edit interface Delegation: change dialog data
Model Data Observer	retrieves data from the *Dialog Data Model* after being notified as observer of that model, loads data for *Presentation*	Referenced: Dialog Data Model read interface, Model Data Observer interface	Implementation: Model Data Observer interface Delegation: read dialog data

With respect to *View State Changes*, the Quasar client reference architecture [16] seems to miss an interface provided by *Presentation* that can be called by the *DialogKernel* to trigger changes like the activation of a dedicated panel that displays properties when the user performs a certain selection. Reference [14] states that this problem can be solved via an additional observer pattern instance.

GUI Framework. As far as the *GUI Framework* is concerned, we decided for the distinction of layout and UI-Control specific knowledge or types. The *UI-Control Library* implements all operations and types that are required for the instantiation of any available UI-Control, the modification of its properties (*UI-Control Properties*) and the definition of its data content (*Technical Data Models*). Often there are various data types with different complexity associated to the available UI-Controls of a framework. They need to be handled by the *Presentation Data Handling* category in order to store and retrieve data in the specific formats like lists, trees, text areas or table grids.

The applied branching of the *GUI Framework* serves the fine-grained presentation of dependencies, so that these model what detailed relationships the other software categories have with *T* software categories.

Dialog Logic. The last main category that is to be placed in the vicinity of a *Dialog* is the *Dialog Logic*. Software categories that are involved in the data structure definition and its logical processing refine the *Dialog Logic*. The basis of these categories is provided by the Quasar client [2][16] and the *Model* part of the MVC pattern [18]. In analogy to the *Presentation* category, we distinguish the definition of data objects (*Dialog Data Model*) with associated operations and the event handling (*Dialog Event Handling*). The latter are based on *Dialog Data Model*, since dialog state evaluations largely depend on current *Dialog Data Model* states, which already reflect the inputs and choices the user may has actuated.

Dialog Data Model. The software category *Dialog Data Model* depends on knowledge about the *Domain Data Model* defined by the *Application Kernel* as well as *Data Queries* that may deliver the composition of selected attributes from different entities in order to create new aggregates relevant for display. The *Data Queries* category belongs to the *Application Server Calls* category, which encapsulates knowledge about the available application services, their pre-conditions, invariants and possible results with respect to the presentation logic layer (see Section III.C.3)).

The *Dialog Logic* category graph mostly constitutes pure *A* category refinements. However, the *Data Conversion* category is of mixed character. To define data structures that can be used in close cooperation with the *Application Services*, it needs to know about *Dialog Data Model,* and thus, incorporates its dependencies to the *Data Queries* and *Domain Data Model*. Besides, the *Data Conversion* category has to be aware of the current *Technical Data Models* in order to provide access for *Presentation Data Handling*. The

latter has to know about the structure of defined data models (*Dialog Data Model* and *Technical Data Models*) to be able to delegate proper updates in both directions.

TABLE V summarizes the responsibilities that are concerned with handling *Dialog* data.

TABLE V. DIALOG DATA MODEL SOFTWARE CATEGORIES SUB-TREE.

Sub-Category	Topic	Contained entities	Operations
Dialog Data Model	establishes the data model used in the entire *Dialog* unit, serves as a global *Model* element according to MVC terms	Defined: Dialog Data read interface, Dialog Data write interface, aggregates or selections of business objects and their attributes (intermediaries [31]), additional data evaluation or interpretation information not present in *Domain Data Model*, list of observers Referenced: data read and write interfaces of *Domain Data Model* and data types (*Data Types and Validation Rules*), Presentation Data Handling interface (observer update), Dialog Data Model interface	Delegation: data read and write operations on the *Domain Data Model* and its data types, notifies *Presentation Data Handling* about data changes (observer pattern) Algorithm: offer browsing and selections of contained *Dialog Data Model* elements specific for display choice options Implementation: Dialog Data read interface, Dialog Data write interface, Dialog Data Model interface
Data Validation	validates *Dialog* data This responsibility may cover the comprehensive validation of multiple attributes or objects at once. Otherwise just the validation interface of individual objects is called and evaluated in order to provide validation information for the *Presentation*.	Defined: validation information beyond the scope of single business objects or data types Referenced: validation interface of *Domain Data Model* or its data types (Data Types and Validation Rules), Dialog Data write interface	Creation: validation information Algorithms: validation of *Dialog* data objects beyond the scope of single objects Delegation: call the validation interface of *Domain Data Model* or its data types Control: Change *Dialog Data Model* based on validation results
Data Conversion	offers transformations between technical and domain data model formats The *Dialog Data Model* may define new getters and setters that accept GUI *Technical Data Models* types or may trigger the call of a dedicated component (R software) [6] providing generic conversions.	Referenced: data read interface of *Domain Data Model* and data types (*Data Types and Validation Rules*) (derived from parent category *Dialog Data Model*), *Technical Data Models*	Algorithm: data conversion operations Delegation: data read operations from both data model formats

The *Dialog Data Model* serves as the primary *Model* according to MVC terms; UI-Controls do only hold their properties that mirror small parts of the *Dialog Data Model*. Furthermore, observer functions are considered *0* software and can be included anywhere, so they require no special interfaces. For the sake of completeness, selected operations have been included in TABLE IV and TABLE V.

Dialog event processing. The entire event processing chain and its association to software categories was challenging; our rationale will be explained as follows.

Foremost, logical and presentation states were separated: presentation logic tends to be stable (enter data, evaluate, present suggestions, make a choice and confirm), is traced to functional requirements (see Figure 9), and thus, should be decoupled from GUI layout specifications. Although the flow of presentation logic is unaffected, the GUI and its technology supporting the user in his tasks may be altered several times starting with updated visual specifications and ending with the deployment of different *GUI Frameworks*. Additionally, the *Presentation* can be further differentiated into abstract visual states that have a close connection to the current application state (or *Dialog Data Model* of Figure 9) and technological or concrete presentation states, which implement the former by using visual appearances. The latter is translated to GUI UI-Controls via *GUI Framework* and its sub-categories. As result, we identified three major categories for state control to be considered below.

The *Dialog Event Handling* tree governs the presentation logic part of a *Dialog* and has no concrete visual representations or related tasks. In contrast, it assumes the *Presentation* to maintain appropriate visual representations, but these remain abstract for the *Dialog Event Handling*, e.g., a *View* for data input is activated, data input was completed or current data leads to another *View* state for data input.

The responsibilities for dialog event handling and respective software categories are summarized in TABLE VI.

TABLE VI. SOFTWARE CATEGORIES RESPONSIBLE FOR DIALOG EVENT HANDLING.

Sub-Category	Topic	Contained entities	Operations
Dialog Logic	The software category and its children are responsible for the presentation logic part of a *Dialog* that connects application logic and presentation technology. Defines interfaces used in child software categories for construction purposes.	Defined: Dialog Data Model interface, Dialog Event Handling interface, Dialog Logic Construction interface	Abstract
Dialog Logic Construction	constructor of a *Dialog Logic* unit	Referenced: Dialog Data Model interface, Dialog Event Handling interface, Presentation Construction interface, Dialog Logic Construction interface	Creation: *Dialog* units with their comprising parts of *Dialog Data Model*, *Dialog Event Handling*, *Presentation* (initial state of a *Dialog* is created) Implementation: Dialog Logic Construction interface
Dialog Event Handling	definition of *Dialog* states and associated actions It is computed what actions are allowed (reload data, confirm) in a given *Dialog* state and how the *Dialog* is altered because of received events. The results or reactions of the *Dialog Event Handling* are each modeled by child software categories: *Dialog Lifecycle Actions*, *Application Server Calls* or *Presentation State Updates* are activated, which enable different behavior or control states of other lower situated entities (*sub-dialogs*, *follow-up dialogs*, *Presentation*). However, the parent category *Dialog Event Handling* resumes the task to decide what child category is finally called in a certain *Dialog* state. In some *Dialogs* data evaluations are needed to trigger the proper *View* from several configurations, which may be rule-based. In this regard, the logic required for changing pages in large scale *Dialogs* like wizards when data was validated successfully is modeled by this software category. The evaluation is done by the *Dialog Event Handling*, but the actual change of *View* is performed by *Presentation State Update*. The latter receives the command to just switch to a certain *View*. The decision to what view is to be switched lies in the scope of *Dialog Event Handling*. Please note that the branching of *Views* is not assigned to the *Dialog Data Model*, since the model can be reused elsewhere with different rules for navigation or display.	Defined: dialog state model, dialog event forwarding interface, dialog event reaction interfaces (Dialog Lifecycle Actions interface, Application Server Calls interface, Data Queries interface, Presentation State Update interface) Referenced: Dialog Event Handling interface, Dialog Data Model	Creation: dialog state model Algorithm: evaluate current *Dialog* state and determine appropriate reactions (e.g., evaluate *Dialog* state on the basis of *Dialog* data in order to determine navigation options) Implementation: Dialog Event Handling interface (constructor), dialog event forwarding interface (called by *Presentation Event Handling* to notify about events to be processed) Control: Call appropriate event reaction interfaces, proper sequences of *Application Server calls* or *Dialog Navigation*
Dialog Lifecycle Actions	construction of *Dialog* units, changes global states of current and other *Dialogs* The scope of this category is the reaction on special events like OK, Cancel and similar terminal notifications. As a result, an entire *Dialog* unit is created or discarded. The associated design units represented by *Dialog Data Model* and *Presentation* are created indirectly by activating a cascade controlled by the *Dialog Logic* and its states. In addition, other *Dialog* units may be ordered to be activated or de-activated by calling the *Dialog Navigation* interface.	Defined: Dialog Navigation interface Referenced: Dialog Logic, Dialog Lifecycle Actions interface, Dialog Logic Construction interface	Creation: Dialog Logic creation / deletion (*Dialog Data Model* and associated *Presentation* are created or deleted implicitly) Implementation: Dialog Lifecycle Actions interface (called by *Dialog Event Handling*) Control: determines the proper sequence of *Dialog* units to be activated and de-activated (Dialog Navigation interface)
Dialog Navigation	performs the navigation among *Dialogs* or activation of *Sub-Dialogs* The opening and closing of auxiliary *Dialogs* like search dialogs for master data (e.g., customer ID and address) is performed.	Referenced: Dialog Navigation interface, Dialog Data Model, Dialog Logic Construction interface (other *Dialog* instance units)	Create: Create and discard *sub-* or *follow-up dialogs* Implementation: Dialog Navigation interface
Dialog State Changes	addresses the possible changes in state with respect to the currently active *Dialog* only	Abstract	Abstract

Sub-Category	Topic	Contained entities	Operations
Application Server Calls	event handling routines that interact with the *Application Logic* services This software category models the reactions on particular events that require the activation of services of the *Application Logic*.	Referenced: Application Server Calls interface, Application Services interface, Domain Data Model	Implementation: Application Server Calls interface Delegation: Application Services interface
Data Queries	loading and updating domain layer data As a specialization of *Application Server Calls*, the retrieval and sending of data in correspondence with the interfaces of *Application Services* is of particular interest.	Referenced: Data Queries interface, Application Services interface, Domain Data Model, Dialog data interface	Algorithm: Assembly or selection of appropriate data queries provided by *Application Services* Implementation: Data Queries interface Delegation: Proper calling sequence of *Application Services* for data retrieval Control: Setting *Dialog* data
Presentation State Update	triggers the change of *Presentation* states / visual layout	Referenced: Presentation State Update interface, View State Change interface, View Navigation interface	Implementation: Presentation State Update interface Delegation: calling of state change notifications of the *Presentation* (View State Change Interface, View Navigation interface)

The interfaces that connect the software categories for event handling are to be defined in detail as reusable *0* or *A* software (much like the observer pattern [17]). That is why there are no dependencies visible in Figure 10 between *Dialog Event Handling* and *Presentation Event Handling*. The same applies for the visibility between *Presentation State Update* and *View State Changes* or *View Navigation*. Finally, a command [17] interface may be used that contains only stereotype operations and can be typed as *0* software. Each of the involved event handling software categories is implicitly connected to *0* software via the various parent software categories in the hierarchy.

Please note that the parent software categories of *Dialog Event Handling* and *Presentation Event Handling* define most interfaces for their children, so that they are able to control them but do not depend on their detailed actions, internal types or implementations. The children encapsulate the results of a response chosen by the parent category for a certain event.

From the presentation logic's perspective, a *Dialog* may adopt different states during runtime. The required knowledge to enact these states is represented by the abstract category *Dialog State Changes*: only its refinements will be assigned to design units; the parent software category *Dialog State Changes* serves grouping purposes and summarizes commonalities of the children. *Dialog State Changes* is separated into children, which either interact with the *ApplicationKernel* or the *Presentation*. Both its categories reflect the two general situations that may occur in any *Dialog*: *Application Server Calls* may be initiated or a *Presentation State Update* can be triggered. The parent category *Dialog Event Handling* possesses the knowledge how to react in a given situation. Its children are dedicated to solely apply the required change of state that either addresses the *Application Server* or *Presentation*, which provide the state change execution. Thus, the children and other server-like entities (e.g., *Application Services*, *View Navigation* and *State Changes*) do not know when their services are called.

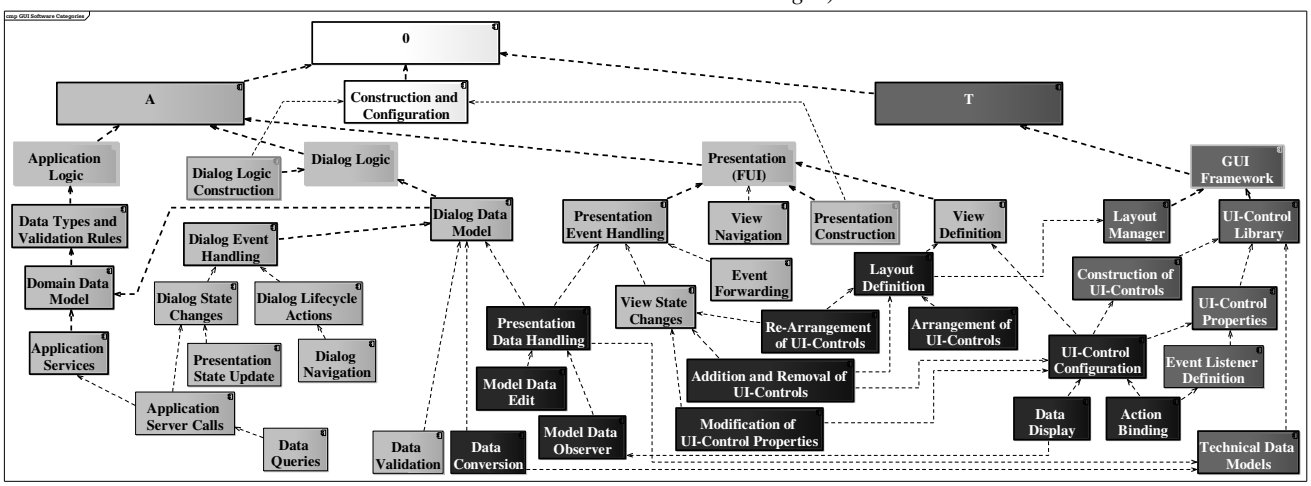


Figure 10. GUI responsibilities modeled as a software category tree.

4) Object Lifecycles and Construction

In this Section, we briefly describe how the construction of instances is considered by the software categories of Figure 10.

As we learned from Figure 9, there are the principal GUI design units *Dialog*, *Sub-Dialog* and associated *Presentations*, which will bear the major part of responsibilities in real GUI systems. To lead to creation of these units, we have incorporated constructor responsibilities within the software category tree that compare to the *DialogManger* of the Quasar client (see Figure 3). Particularly, the *Dialog* and *Presentation* both were supplemented with responsibilities dedicated to construct the child elements of these parent software categories.

In this regard, the *Dialog Logic Construction* is responsible for the creation of the main *Dialog* unit. We assume that a *Dialog* design unit will correspond to the software category sub-tree modeled by *Dialog Logic*. Based on the responsibilities a *Dialog* has to fulfill, it initiates the construction of the starting *Presentation* as an entry point for user interaction after the creation of own member objects. This sequence is to be followed, since the *Dialog Logic* controls the states of the *Presentation* anyway.

Concerning the *Presentation*, this design unit also features a software category (*Presentation Construction*) dedicated to the construction of its child elements.

Both the *Presentation* and *Dialog Logic* may call the construction of additional units of their type when respective events occur: for the *Presentation*, new *Views* will be requested by *View Navigation* upon a call from *Presentation State Update* was received. With respect to the *Dialog Logic*, during the event processing by *Dialog Event Handling* a *Dialog* may be finalized or a new *Dialog* instance may be created as a result of a *Dialog Navigation* event reaction. Both options are controlled by *Dialog Lifecycle Actions*.

Figure 11 provides an overview about the dependencies concerning lifecycles and construction of instances based on the software categories of Figure 10.

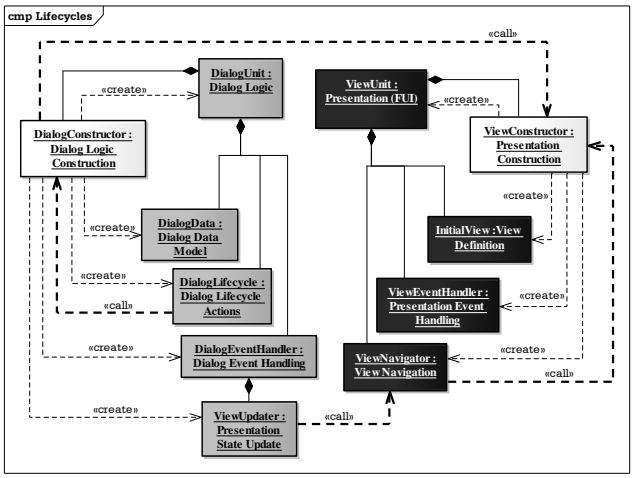


Figure 11. Intended lifecycle dependencies and constructors of possible objects derived from the GUI software category model.

Whenever new instances are to be created, an object that implements the respective construction responsibility of either *Dialog Logic Construction* or *Presentation Construction* is to be delegated.

5) The Event Processing of the Software Categories

Figure 12 provides an overview of possible interface connections between software categories involved in event processing. Please note that the interfaces need to be of the basic *A* category type as this is the common parent category of the displayed interacting categories. Basically, three different scopes for states are modeled by the software categories. They are the following:

- *Dialog Logic - Application Services*: The scope of this state model is concerned with the data model of the entire *Dialog* unit and the interaction with *Application Services*. Decisions are to be taken what services and data contents are to be combined for the required interaction of a given *Use Case*. As a result of the *Dialog Logic* state model evaluation, a change of the visual state may need to be delegated. It depends on the GUI specification with respect to the required steps a given *Use Case* scenario may demand for.
- *Dialog Logic - Presentation*, *View* level: A *Dialog* may require consecutive *Views* to be displayed in a certain sequence or based on user decisions. These changes of *Views* are in the scope of a dedicated state model.
- *Presentation*, UI-control level: The different states a particular *View* may adopt are considered herein. This covers different changes in layouts and UI-Control configurations.

The general flow of events is indented to work as follows: initially, the user triggers some events that are intercepted by UI-Controls that have an *Action Binding* configuration. In any case, the event is passed on via *PresentationEventHandlingInterface* to the *Presentation Event Handling*. A first evaluation of that event may result in a decision by *Presentation Event Handling* to further move the event on the event processing chain via *EventForwardingInterface* to *Dialog Event Handling* for the final evaluation.

Depending on the current state of the *Dialog*, *Dialog Lifecycle Actions* (creation and deletion of *Dialogs* and their objects), a *Dialog Navigation* (change of current *View* or the instantiation of *Sub-Dialogs*), *Application Server Calls* (commit a sequence of service calls) or a *Presentation State Update* (change of the visual representation) may be activated as reactions by *Dialog Event Handling*.

In this regard, the key design issue is that the *Presentation* has no knowledge in its sub-categories how to decide on a proper reaction for events relevant for *Dialog Logic*. Please remember that *Presentations* or *Views* may be reused in different contexts (compare pluggable *Views* in reference [31]), and so, a direct binding of their UI-Control events to state changes would greatly limit their flexibility and adaptability. Therefore, the event firstly is forwarded via

the *DialogEventHandlingInterface* interface of Figure 12. Then, the *Dialog Event Handling* evaluates the event and controls one of its children, which further delegates to the displayed interfaces of Figure 12 and initiates the final change of state. Concerning the *Presentation State Update* in Figure 12, *View State Changes* (panels are activated) or *View Navigation*s (wizard steps or tabs are switched) are committed via interfaces. Another option would be a change of the *Dialog's* lifecycle or even a *Dialog Navigation* (separate *Dialogs* or an auxiliary search *Dialog* are instantiated) could be performed.

In this context, the knowledge when to trigger any of the interface operations is kept in the parent category *Dialog Event Handling*. In contrast, the execution of the respective state change is encapsulated in the child categories, which are marked by a white border in Figure 12 and implement the interfaces. At last, the state changes are completely decoupled from the point in time when they are requested.

Moreover, the *Presentation Event Handling* is separated into event processing that is either concerned with data or the visual structure. Mostly the data relevant events can be processed locally by the *Presentation* if no forwarding is registered. However, the *View State Changes* do require the forwarding of events to the *Dialog Event Handling* first, before they can be committed. This is due to the decoupling of *View* states and their better exchangeability. Furthermore, the differentiation of event evaluation, triggering and state change execution supports the reuse and change of *Views* as they are better decoupled from *Dialog Logic* components. In this regard, *View* states are relevant for the *Dialog Logic* but not their concrete appearance, which can be adapted frequently.

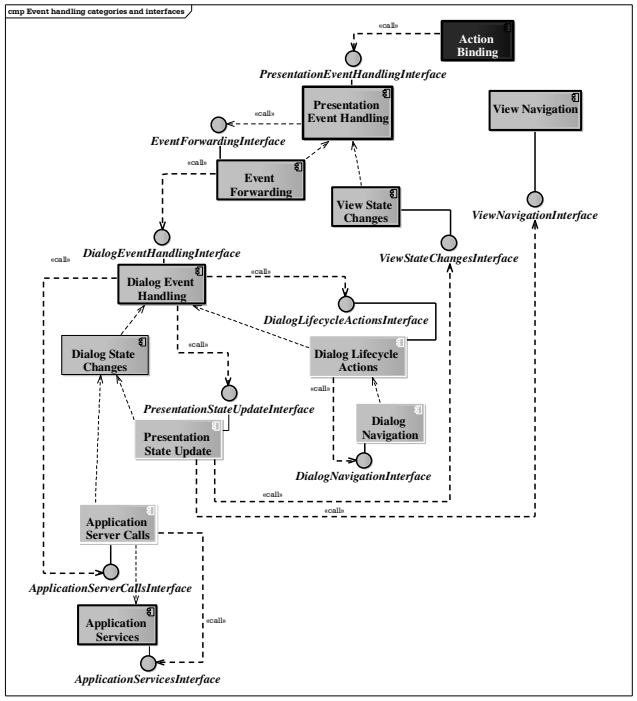


Figure 12. Software categories relevant for event processing and possible interfaces.

To conclude, the event handling approach and its respective software categories ensure that the layers of presentation technology and logic (introduced in Section III.C.3)) remain strictly separated. In fact, there will be dependencies among *Dialog Logic* and *Presentation* that cannot be avoided like the consistency of logic and visual states. However, the control of all states remains centered in one unit of design (*Dialog Logic*), which will facilitate development and maintenance of complex *Dialogs*.

IV. Review of GUI Architecture Patterns

In this section, we review the presented GUI patterns of Section II in the light of the elaborated software categories.

A. MVC Variants

For the review of classic GUI architecture patterns, we would like to refer to exemplary work published in [4] and [10], which is valuable for filling gaps and giving directions for related design decisions. Therein, options for refinement and customizing MVC based architectures are proposed and discussed. It is still up to the developer to decide on the several choices. In contrast, the Quasar client architecture presents a reference for our domain that already has some refinements incorporated.

1) Positive Aspects

Both patterns and Quasar client share two positive aspects that motivate their application. Firstly, the data storing component does not depend on any other of the components, and so, can independently evolve. Secondly, only one of the components resumes the task to call *ApplicationKernel* services. This aspect eases the design efforts for interfaces and data exchange formats between *Dialogs* and *ApplicationKernel*.

2) Issues

According to the MVC variants, we see major design issues that will be described in the following paragraphs.

Separation of concerns. To begin with, the degree of encapsulation and separation of concerns of MVC variants is very limited. There is no variant that is able to reduce the dependencies of all three abstractions altogether. Solely, the distribution of tasks is altered, and so, the visibility among components changes accordingly. In the end, one component will be assigned responsibilities that originate from the two other components as they are defined by classic MVC [31]. Therefore, the component with concentrated tasks tends to be overburdened, and finally, can end up as the bottleneck from a maintenance perspective. Additionally, altering the tasks of the three components in certain variants may result in a simplification of one component that can only be employed for stereotype tasks but fails to suit more complex scenarios. There seems to be no ideal separation of concerns among the three components. A fourth element may be missing.

In general, there are no hints given how the display for certain portions of business logic or data can be decoupled from their technical manifestation. More precisely, the *View* part is directly coupled to the *GUI Framework* (Figure 1). In addition, the knowledge of the *View* has to constitute of how to operate the *GUI Framework* facilities (to construct the visual dialog parts) and what layout as well as what

selection, order and arrangement of UI-Controls are needed to embody the domain and the current service in use.

Event differentiation and related control. With regard to the event processing chain of Section II.B, the GUI patterns do not distinguish clearly between events related to technical or application concerns. In general, a guideline is missing for the decision when to shift between presentation technology or presentation logic related processing of events. TABLE I provides an overview about the assignment of these layer specific responsibilities to MVC pattern roles.

Although the MVP variants [6][7] and HMVC [5] employ a "Supervising Controller" [15], which receives each event from any UI-Control and acts as a global MVC *Controller*, the problem persists: the *Presenter* as well as the HMVC *Controller* still have to decide whether the incoming events require an presentation technology or presentation logic specific processing and have to react accordingly. Yet, these approaches solve the "visibility problem" described by Karagkasidis [10] where the *Controller* and *View* are separate classes. In any case, the developer has to refine the architecture by himself to enable a differentiated handling of presentation and application related events. Finally, the reuse may be affected, since the *Controllers* end up processing both types of events for the sake of initially quick releases.

Cohesion and granularity of triads. With the application of MVC derivates that differ from the classic MVC approach [31] a problem occurs concerning the identification of possible instances and their proper size. There are hardly any hints when to create new *Dialog* instances or MVC-triads. Thus, the proper modularization of *Dialog* components is to be done on behalf of the developer. Only the HMVC [5] gives some rudimentary hints. In the end, the general size and scope of MVC triads is not clear. According to Karagkasidis [10], a *View* may constitute of single UI-Controls (widgets), containers like panels with a certain set of UI-Controls or complete *Dialogs*. The classic MVC approach [31] was clearer on that topic, since MVC triads were very fine-granular starting at the UI-Control (widget) level and building a corresponding triad for every element of the visual object hierarchy, ultimately ending with a last triad at the window level. However, the classic approach is not likely to be feasible for modern and more complex application scenarios: the high integration of business systems and their complexity would demand for a large number of *Dialogs* that would result in myriad of MVC triads.

Coupling of *Controllers* to both *Model* and *View*. With respect to the above described limited separation of concerns more issues arise. The controlling of both *Presentation* states and the handling of application related events to initiate *ApplicationKernelService* calls inside the *Controller* creates close coupling of *Controllers* to both *View* elements and naturally the *Model*. Usually, in many MVC variants *Controller* and *View* maintain a strong dependency where the *Controller* is fully aware of the UI-Controls of the *View*. In fact, both components build an aggregated unit of design (rather than representing separated classes) that cannot be reused and is harder to maintain. Eventually, a *Controller* can only interact with *Views* that comply with a certain set of states. Whenever the set of UI-Controls changes the possible states of the entire *Dialog* alter as well, so that the *Controller* implementation may have to be revised each time. This is due to the awareness of *Controllers* about the *View's* UI-Controls what results from the following. In modern GUI frameworks the *Controllers* obtain user entered data directly from UI-Controls and not as the payload of an incoming event, as this was the case in Smalltalk or classic MVC [31]. With the latter, separate classes for *View* and *Controller* could be realized but current GUI frameworks demand for alternative solutions. Karagkasidis [10] exemplarily discusses the solution provided by HMVC.

To partly resolve this issue and decouple the *Controller* at least from application aspects, a developer could revert to the "Model as a Services Façade" [4] MVC variant. The *Model* would be assigned both data structures and related service calls for interaction with the *ApplicationKernel*. This step would raise a comparative discussion as whether it is favorable to build a separate service layer [44] or use the domain model pattern [32] exclusively for the structuring of the *ApplicationKernel*. In our opinion, the *Model* should not act as a service façade, since it would make parts of an *ApplicationKernel* service layer obsolete. According to the resulting dependencies to functional requirements, the traceability-links of *Use Cases* or tasks would be scattered among different *Models* and parts of the *ApplicationKernel*. Furthermore, the operations of the *Model* would be closely coupled to a certain data structure so that a *Model* cannot be easily combined with other application services in the future. Lastly, services should prevail, since there might be other clients besides a particular GUI to rely on services. There are more disadvantages with that solution like the stereotype character of the *Controller* [4], which will only serve a certain pattern of interaction. Thus, the *Model* should only contain data-relevant operations (getter, setter, aggregation and conversion, a state of current selections, validation state) and be reusable with other services. In this regard, the *Model* should act as a mere preparation of a data structure that is useful in the context of a *View*, its display, as well as in- and outputs.

3) Summary

The MVC and its derivates require much adaptation in order to be prepared for implementation [14]. The above mentioned issues may considerably have a negative impact on the resulting architecture quality. The available patterns are definitely not easy to interpret with respect to the common set of GUI responsibilities illustrated by the software category tree in Figure 10.

The tracing of functional requirements to the parts of the GUI, which coordinates *ApplicationKernel* service calls, will largely depend on the refinements the developers have incorporated in the GUI architecture. Additionally, a clear separation of presentation technology and logic (see Section III.C.3)) is not supported in any variant, so that event handling will always consume high efforts for development and especially maintenance.

Anyway, the resulting architectures will be heterogeneous and may add complexity to quickly provide an adapted solution for the particular domain. As long as there are no standard architectures or standardized

responsibilities available, the developer is left with many choices that potentially will lead to vast differences in software architecture quality. The improved segregation of software categories in component architectures is a challenging goal hard to achieve with available patterns. Project budgets may severely limit the software architecture quality to be attained.

B. Quasar Client Reference Architecture

1) General Valuation

The Quasar client architecture provides the most detailed architecture view on GUI systems published so far and can be regarded as a refinement of the common GUI patterns.

Positive aspects. In contrast to the MVC variants, the Quasar client separates *Presentation* and *DialogKernel* as principal dialog components. This separation is the main source for its virtues, since more clearly distinguished *Controller* tasks are achieved. In this regard, the *Presentation* is required to handle technical events and the *DialogKernel* will process application related events in close cooperation with the *ApplicationKernel* services.

States and control. According to Siedersleben [16], the *Presentation* and *DialogKernel* components share a common structure: both possess memory for storing data, states and a control. Thus, both components are able to manage their states independently. A change of layout aspects in the *Presentation* would not affect the *DialogKernel* accordingly.

In theory, the changes of states are implemented in each component individually and can be triggered by *A* typed interfaces that may be designed on the basis of a command [17] pattern [14]. Consequently, the *DialogKernel* does not require knowledge about the inner structure of the *Presentation* and vice versa. Thereby, the *Presentation* may provide a set of operations that alter the layout of a *Dialog* depending on the current content of data received from the *DialogKernel* via *DataUpdate* interface. The triggering of visual state changes on behalf of the *DialogKernel* (*Presentation State Update*) may be possible that way but is not considered. For instance, a *DialogKernel* was notified via *DialogEvent* that the user has selected an item in a table listing available products. But the product is on back-order, so the *Presentation* should receive the command to display a certain state of the button bar, e.g., deactivate the "add to cart" button. According to Siedersleben [16], the states of visual elements are exclusively controlled by the *Presentation*. However, in the particular example only the *DialogKernel* would possess the knowledge when to trigger the state change of the *Presentation*. It seems that the cooperation of both units of design needs further elaboration to be able to be implemented in practical examples. Besides, a *DialogKernel* could be able to coordinate the inputs of a user working with two *Presentations* simultaneously.

2) Traceability-Links to GUI Software Categories

To be able to better valuate the Quasar client architecture, we traced the identified software categories of Section III.D to its structural elements. Figure 13 displays the resulting traceability matrix. The sources for traceability-links constitute software categories of varying detail arranged on the left hand side.

	ApplicationEvents (AE)	ApplicationEventsRegistration (DA)	ApplicationKernel	ApplicationKernelService (AF)	DataRead (R)	DataUpdate (SY)	DialogActivity (U)	DialogCompletion (V)	DialogEvent (DE)	DialogKernel	DialogManager	InputDataQuery (A)	Presentation	PresentationEvent (PE)	SessionControl	ViewDefinition (DP)
Action Binding													↑	↑		↑
Addition and Removal of UI-Controls													↑			↑
Application Server Calls			↑	↑						↑						
Application Services			↑	↑												
Arrangement of UI-Controls													↑			↑
Data Conversion													↑			
Data Display													↑			↑
Data Queries			↑	↑						↑						
Data Types and Validation Rules			↑													
Data Validation			↑							↑						
Dialog Data Model										↑			↑			
Dialog Event Handling										↑						
Dialog Lifecycle Actions							↑	↑			↑					
Dialog Logic Construction											↑					
Dialog Navigation							↑	↑			↑				↑	
Dialog State Changes	↑	↑								↑						
Domain Data Model			↑													
Event Forwarding									↑							
Layout Definition													↑			↑
Model Data Edit									↑	↑		↑	↑			
Model Data Observer					↑	↑				↑			↑			
Modification of UI-Control Properties													↑			↑
Presentation Construction											↑					
Presentation Event Handling													↑			
Presentation State Update													↑			
Re-Arrangement of UI-Controls													↑			↑
UI-Control Configuration													↑			↑
View Navigation																

Figure 13. The GUI software categories traced to Quasar client reference architecture components and interfaces.

Please note that the general parent software categories were excluded, since all child categories are presented in the matrix. On top of the matrix, the traceability-link targets are represented either by the components or interfaces of the Quasar client. Components not relevant as traceability-link targets were excluded.

Interpretation. We need to provide directions about the treatment of interfaces and connected dependencies, which are depicted in Figure 3. A client that imports and calls a foreign interface must have knowledge about the proper usage and sequences of operations. In fact, the deeper and more chained the commands (compare delegation and control of Section III.C.4)) are the more likely is the mixture of software categories. Finally, the client will be dependent on the same software category the interface is composed of.

This particularly applies to the *Presentation* (obviously an *AT* component) that extensively uses the *GUI Framework* interfaces, which are to be included in the traceability matrix.

In contrast, single commands of abstract or stereotype nature like notify calls can be realized with a *0* type interface. Yet, the interfaces pose hard to valuate concepts as they inspire a dynamic view on the architecture like the sequences of commands or flow of algorithms. Ultimately, the interface operations would need further refinement for a final valuation. Partly, the Quasar reference architecture provides basic sequences for interfaces in [2].

Separation of concerns. For the valuation of both cohesion and separation of concerns two directions inside the traceability matrix of Figure 13 have to be considered.

Horizontal. The horizontal direction displays a number of marks for the realization of software categories though components or interfaces. For a high cohesion and well separated concerns, there should be software categories realized only by components or interfaces that belong to one unit of design. In sum, *Application Server Calls*, *Data Queries*, *Data Validation*, *Dialog Lifecycle Actions*, *Dialog Navigation*, *Model Data Edit* and *Model Data Observer* are realized by several Quasar elements, and thus, different units of design.

The first three software categories mentioned before are shared among the *ApplicationKernel* and *DialogKernel*. Thus, the resulting coupling between these design units will largely depend on the refinement of interfaces between both components. Eventually, a mixture of *A* software categories can be a probable result when no *0* interfaces can be invented. The details of this client and server communication remain an open issue as well as the construction of *Data Queries*.

Besides, *Model Data Observer* is presented with two options that are either implemented by the *DialogKernel* (*DataRead*) or *Presentation* (*DataUpdate*). However, the complementary task of *Model Data Edit* is only briefly mentioned. Siedersleben states that the *Presentation* may know the *DialogKernel* and its data interface (see association in Figure 3) but not vice versa [16]. As an alternative, newly entered data may be included as payload of the event emitted via *DialogEvent* by the *Presentation* [16]. How the important task of changing dialog data is performed in detail by the *Presentation* and what interfaces are required is finally left open.

Moreover, *Dialog Lifecycle Actions* are of less importance. They are rather stereotype operations that could be detailed by *0* type software. In contrast, for the *Dialog Navigation* there may be missing directions in the Quasar client reference architecture, so that responsibilities have to be refined on behalf of the developer. We wonder how dialog sequences resulting from task model specifications [45] would affect the software category assignments. Maybe the *Session* cannot be marked as *0* software anymore, since it would need knowledge of the proper sequence of dialogs, and thus, would be designated as *A* software that could not be reused for different task model instances.

Vertical. A further assessment considers the vertical direction that reveals targets with many traceability-links. This can be a marker for lacking detail or even low cohesion. Those targets would take on too many responsibilities at once. There are multiple candidates that awake our attention.

As already stated above, the *ApplicationKernelService* needs further refinement, so that the way how calls and data queries are performed by the *DialogKernel* are both detailed and differentiated concerning allowed data types and resulting coupling. Consequently, another major issue is the *DialogKernel* itself. This component is relatively vague in definition, so that tasks like calls to the *ApplicationKernel*, *Data Queries*, the *Dialog Data Model* definition, *Data Validation* and the control of states need to be elaborated from scratch.

Concerning functional requirements tracing, the *DialogKernel's* internal structure and state control are important issues that affect the resulting dependencies to requirements. For instance, it has to be decided what portions of a use case will be exclusively realized by the *Application Services* and what parts the *DialogKernel* is in charge of. Above all, the *DialogKernel* is likely to depend to some considerable extent on the *ApplicationKernel* and its *Domain Data Model*. In this regard, it has to be cleared how *Data Queries* are to be handled from the *Dialog Data Model's* point of view. The *Dialog Data Model* can either be composed of pure entities, which may be embedded as interfaces or data transfer objects, or aggregations that are sourced from selected attributes of several entities retrieved by a query.

Furthermore, the *Presentation* also requires further elaboration in design. Being the complementary part of the *DialogKernel* in a *Dialog*, the *Presentation* is declared as having its own data model in parallel to the *DialogKernel* in order to perform conversions to the *Technical Data Models*. The main data definition is assigned to the *DialogKernel*, since this component is in charge of any data retrieval from the *ApplicationKernel*.

How the data related communication (read and edit) besides the notification of updates between *Presentation* and *DialogKernel* is originally intended remains another open issue. In this regard, design decisions on both interfaces and data types as well as their connection to the *Domain Data Model* have to be considered. Moreover, details about the triggering (*Presentation State Update*) and execution of *View State Changes* are missing. This is due to the unclear connection between *Presentation* and *DialogKernel*. When decisions about reactions on events are bound to *Presentation*, logical behavior will be closely coupled to certain *Views*, so that they are less flexible for change and reuse. In addition, events can only be emitted by *View* elements and cannot be triggered by the evaluation of gathered *Dialog* data alone, since there is no link for the *DialogKernel* to initiate a *View State Change* via *Presentation State Update* when an event was forwarded.

A look at the matrix of Figure 13 reveals that the event handling of the Quasar client architecture with respect to presentation technology and logic concerns seems not to be elaborated with the necessary care and accuracy; there are several responsibilities mixed within and among *Presentation* and *DialogKernel*: firstly, the *Presentation* is in charge of both receiving events (*Presentation Event Handling*), deciding on visual states (*Presentation State Update*) and executing them (e.g., *Addition and Removal of UI-Controls*). Secondly, the needed knowledge for decisions, and thus, presentation logic is likely to be based within the *DialogKernel* as far as the interaction with the *Application Services* is concerned. Yet, the latter is assigned to handle its own state model (*Dialog Event Handling*) and partly manages the *Dialog* data (*Dialog Data Model*) together with *Presentation*. So, both design units share the information necessary for deciding upon state changes. In contrast to the GUI software category model of Figure 10, the Quasar client

architecture assigns state decisions and executions based on a different point of view: presentation logic is strictly separated between application (*DialogKernel - Application Server Calls*) and visual behavior (*Presentation - Presentation State Updates*), so that the *Dialog Logic* and its state model is not centered but shared among two design units. For that reason, with the Quasar client a *Dialog* will be harder to adapt to a changed *Use Case* scenario affecting the *Dialog* state model (a new step with a new or updated *View* is required), since the *Presentation* is designed to both manage and execute the *View* state changes. So, the presentation logic required for deciding on a change or update of the *Views* is lost and has to be re-implemented whenever the *Presentation* has changed. From our point of view, a centralization of event-based decisions found in the GUI software category tree of Figure 10 would reduce the portions of *AT* software existent in any *Presentation* and could partly facilitate the exchange of *Views*.

As far as the visual part of the *Presentation* is concerned, the *ViewDefinition* interface and related implementations inside the *Presentation* need more refinement. The coarse grained interface is employed for both handling view states and their initial construction. In this context, a developer would have to decide on how the *DialogKernel* may trigger the visual state changes as a result of its own states defined by *Dialog State Changes* and its children.

Lastly, the *Presentation* is assigned quite a are large set of responsibilities, but is the design unit that is not likely to be stable or reusable after technological changes compared to the *DialogKernel*, which does not depend on any *T* software influences.

Missing responsibilities. Responsibilities that were entirely not mentioned with respect to the Quasar client reference architecture, was the *View Navigation*. This task may be confused with *Dialog Navigation*. Siedersleben approaches the architecture of a *Dialog* with the definition of the relevant terms in reference [16], but he does not use them in a consistent way, so that some terms are only mentioned and remain unrelated to the Quasar client architecture itself. As a consequence, the design unit of a *Dialog* remains unclear with respect to the delimitation of other *Dialog* instances, *Sub-Dialogs*, and more urgently, *Presentations* or *Views* of Figure 9 that express the different interaction steps with a user.

3) Summary

Our review of the Quasar client revealed that this reference architecture is more advanced than common GUI patterns. It includes most of the common MVC pattern responsibilities (TABLE I) and adds several additional ones (TABLE II). Besides, its main advantage lies in the division of *Controller* tasks among the *Presentation* and *DialogController*, so a better separation of concerns can be achieved. However, this results in increased complexity concerning the number and type of interfaces to be implemented.

In comparison to other architectural patterns, the Quasar client provides more detail and descriptions that give hints to many design decisions, but these are scattered among several sources [16][29][38][14] only available in German language. There was no comprehensive or updated description published, which would provide the needed implementation details. In the end, the Quasar client remains vague with many important issues to solve by individual design decisions. Nevertheless, we learn from the traceability matrix of Figure 13 that there are already hints, which component is to take on what responsibility. In practice, this would yield only a partial improvement with respect to the common GUI patterns. In reference [2], Haft et al. state that the Quasar client could not be standardized, since most software projects required specific adaptations. The many individual refinements would affect the marking of software categories, so that the purity of them and the separation of concerns may not be maintained as intended. Even the Quasar client assumes that some portions of *AT* software cannot be avoided with conventional architectures relying on invasive frameworks.

To conclude, the Quasar architecture is not suitable for a straight forward implementation. As we see, there are still gaps in the reference architecture and the developer has to incorporate own thoughts in order reach the desired quality architecture. The separation of concerns can be improved with a customized Quasar client architecture, but this largely depends on the skills of the architect. In the end, the Quasar client may be a better, and foremost higher detailed, basis for reuse of architectural knowledge than the MVC variants alone.

V. Results and Discussion

1) GUI Responsibilities Software Category Tree

One of our objectives was to provide a software category tree with separated concerns to describe a complete decomposition of GUI architecture responsibilities.

Software category model. We derived a software category model that structures the dependencies among common responsibilities of GUI architecture design units without being biased towards a certain GUI architectural pattern or framework.

Software category definition and modeling. To be able to model detailed, refined software categories and finally delimit them, we had to invent modeling rules that were not provided in the original sources. We are convinced that these enhanced rules create a solid foundation for modeling responsibilities of software architectures, since the results make sense in our case of a better understanding of GUI architecture patterns and bring us further towards UIP integration.

Compared to the CRC method applied for the GUI patterns in [18], the collaborators of a certain software category are summarized in the second dimension but are further outlined by the association with detailed operations. On the CRC cards, every responsibility of a design unit is noted on one card and there are not details about their relationships to the mentioned collaborators on that card.

Nevertheless, there are not only positive aspects about the software category modeling approach. In fact, there are some weaknesses of the software categories tree display: For instance, there is no hint what elements are actually derived from the dependencies of parent software categories. Generally, there can be all included or referenced entities or

only a sub-set of them considered in the child software category. Some contained entities can even be derived from the parents of a parent category (e.g., *Data Display - Model Data Observer - Presentation Data Handling - Technical Data Models* is an example of such a cascade of dependencies or refinements to be discovered in Figure 10). Moreover, there is also no hint, which parent categories are skipped and will not be considered in child software categories. In most cases *0* software is used and almost never skipped, but along the way up to *0* not all software categories are always considered. Some relationships just model the potential visibility of entities. Maybe the detailed modeling of instance based software category trees can remedy some of these aspects by providing further detail.

Shape of the tree. Concerning the actual shape of the software categories tree, there might be different structures or aggregations possible (intermediate categories) but the final child elements clearly mark the occurring responsibilities. In this regard, is has to be noticed that the software categories displayed here are pure and intended to be well separated. This arrangement of responsibilities is mostly not the case in real systems and designs; the software category tree is an ideal construction.

Software architecture relationship. From our point of view, the different MVC pattern variants are hard to understand with all their facets concerning detailed responsibilities and dependencies on other design units they need to interact with. Often the MVC variants compose of smaller patterns like "Supervising Controller" [15] and "Presentation Model" [15], which are a proof of the ever present complexity of GUI design.

To partly address the complexity issue, the software category model presented in this work aims to display the responsibilities of GUI architectures without favoring certain structuring or role assignment of design units. They are created to provide an overview of the general responsibilities that may occur in GUI systems instead. Architects and developers shall get a guide what tasks are to be fulfilled within the GUI system.

There may be an inherent or obvious structure hidden in the separated sub-trees with *Presentation* and *Dialog Logic*. However, this structure simply emerges from the dependencies of knowledge (modeled by the dimensions of Figure 8), which is required for the different responsibilities. The displayed separation or decomposition of software categories has not to be strictly followed; there is rather high degree of flexibility: the software categories can be distributed differently to design units. For instance, the *Data Conversion* responsibility is often differently solved in designs. Some responsibilities may be omitted when requirements do not demand for them. Eventually, the resulting distribution of software categories to design units determines the final quality of the software architecture.

In this regard, architects can consult this model without the need to be restricted by given designs, their roles and relationships. The descriptions and sources used for the composition of the software category tree are not entirely distracting or misleading, yet they are quite helpful for understand certain designs. But their weakness is that they are already biased towards a certain structure of design or effects to achieve like this was elaborated by Alpaev in [4] for the MVC design options.

Software category refinement level. One may argue that the consideration and segregation of software categories may overburden an architect with additional tasks and he will eventually loose overview due to the management of a set of fine-grained responsibilities. In contrast, the software category tree shall be helpful and not a burden. In fact, the software categories build on the refinement from basic to detailed categories in a hierarchical tree. So, the architect principally can decide on the level of detail he applies for modeling, mapping or assessment of design. In this regard, software categories always group several responsibilities into a family of cohesive entities; children retain the more detailed responsibilities and parents serve as a more general aggregation. In that way, an architecturally meaningful re-composition of GUI responsibilities is created. The architect may pick a certain detail level of the category tree, which ideally resembles a prepared separation of concerns in any case, in order to re-distribute these responsibilities in a new system design. This choice decides whether only basic software categories are used for architecture planning or refined ones are applied instead in order to achieve a much better accuracy for cohesion as well as the evaluation of how well concerns were separated.

Software architecture assessments. Furthermore, the software category model can be of aid for the valuation of the detail, cohesion and separation of concerns of reference architectures or patterns. Section IV.B outlined the principal approach and an example that assessed the Quasar client reference architecture. In sum, the software categories approach can reveal not supported tasks, design units that bear many tasks at once, perfect matches and tasks that are shared among two or more units of design.

In our opinion, the established software category tree is well-suited for GUI architecture assessments: the software categories embody a set union of the responsibilities of many of the common GUI architecture patterns. In the context of GUI design, the software categories resemble different and delimited packages of knowledge, which are used to identify and map components or smaller units of design. Later on, the dependencies among the software categories will lead the design of interfaces between components [16] to achieve a minimum of coupling based on the rules established in reference [16]. Thus, the proper distribution of identified software categories among design units can have an enormous impact on software quality. During assessments, this intended way of identifying design units and delimiting them by assigning distinct tasks to them can be reversed. This enables an evaluation of the rationale the design is based on.

Available architecture patterns cannot provide a comparative view on GUI responsibilities, since they miss some details, are interpreted differently among developers, can be biased towards a certain programming language, and the discussion of their trade-offs is limited to their scope, so that the impact on the general architecture can only be partly valuated. In addition, patterns often need to be combined within a design, so that their different effects depend on the actual combination and their adaptations.

Interface design. When common GUI architecture responsibilities have been identified and systematically analyzed concerning their dependencies, the potential interfaces for communication between components or classes can be derived. According to Quasar [16], an interface ideally should be defined on the basis of a software category that serves as a parent for both software categories to be linked. That way, the least coupling is ensured. Not always can a shared parent software category be found to serve as a basis for an interface between components. This may be due to an improper distribution of responsibilities among design units. As a result, the underlying software category model needs to be revised. Anyway, the identification of design units and their interface structure requires some detailed planning.

Relationship to implementations. The responsibilities modeled by the software category tree can be used to analyze and reflect implementations. According to Quasar references, this is only done on the level of the very basic software categories *0*, *A*, *T* and *AT*. With the now available refinements for GUI architectures, an actual design or implementation can be evaluated concerning the correspondence to software categories. Thus, the cohesion and separation of concerns ca be assessed. The other way around, given implementations may refine the software category tree and it could be practically examined if the visibility is sufficient moving the tree upwards starting from a certain category or if additional dependencies have to be modeled.

Missing concerns. Currently, concerns like user profiles, additional assistance, session management [14] and authorization are not included. In general, terms in the field of GUI architecture are not used uniformly, so we rely on our category model that provides a clear description of tasks. Furthermore, the software categories may be adapted to fit other domains, since the separation of concerns is essential in most software architectures.

Summary. By the application of software categories, the GUI responsibilities to be identified have been ordered and grouped according to their knowledge and purpose, but this was modeled independently from any specific software architecture. The software categories in that role are suitable to represent a set of GUI responsibilities without the need to mention specific data types or operations of certain frameworks. Finally, the way how frameworks are applied shall be adapted to the required set of responsibilities as well as the software architecture based thereupon and not vice versa.

2) Major Issues in GUI Architecture Design

Our first objective was to identify GUI design issues. These issues naturally result from points of improper coupling, non-separated concerns and in general missing responsibilities not modeled by available GUI architectures or patterns. We had to analyze the available architectural patterns, which differ in structure as well as the encapsulation of concerns. Finally, there is no standardized GUI architecture ready for implementation. This is an issue here but also for mobile devices [46]. We analyzed the differences or missing details of presented architectural patterns and identified four major design issues that may have a considerable impact on GUI development and maintenance.

Presentation logic and application control flow. Firstly, a design decision has to treat the question what and how much application logic is being processed by a single *Dialog*, or particularly its *Dialog Logic* or *DialogKernel*. Thus, the coordination and division of labor between dialog and application related components should clearly define what portions of the event processing chain will just be handled by the *DialogKernel*.

As the primary controlling entity of a dialog, the *DialogKernel* acts as a client of the *ApplicationKernel* and its services [16][14]. The architect has to decide how much control flow will be implemented by the client and what operations or services are to be integrated in the controlling object's flow definition. For instance, the business logic can be separated by different layers like services, auxiliary services, domain model entities and data types [47]. The coordination of the various algorithms and delegations, which is essential to achieve the goals defined by use cases, can either be performed by the *ApplicationKernel* or the *DialogKernel* may govern the sequence of service calls and their combination. The so called orchestration of services to realize a certain use case is an option for the *DialogKernel*, since this design unit determines the data structure for user interaction. In this context, the *DialogKernel* directly can react to valid user inputs and may decide on the further processing via services or may even trigger corresponding state changes for the *Presentation*. How the latter is to be designed remains an open issue.

Siedersleben states that the *ApplicationKernel* components constitute of use case realizations [16]. However, these components would definitely be incomplete use cases realizations, since the latter regularly require much user interaction. To conclude, the question arises how use case realizations are sub-divided among *ApplicationKernel* services (management of data structures and relationships, service hierarchy), *DialogKernels* (logic for dialog flow and control of user interaction) and finally *Presentations* (visual part, in- and output UI-Controls, realization of visual states). Ultimately, this design decision depends on the navigation structure and whether one *DialogKernel* may control a composition of *Presentation* units or *Sub-Dialogs* that form a complete *Dialog* unit for the sake of one use case realization.

Dialog navigation. This leads us to the second issue that is concerned with the flow of *Dialog* units or navigation among them. Karagkasidis [10] already described this issue from the perspective of an example with opening and closing *Sub-Dialogs*. Important aspects mentioned by Karagkasidis are the lifecycle management of *Sub-Dialogs* that can be related to our presented GUI design units of *Dialog, Sub-Dialogs* and *Views* from Figure 9: they need to be controlled by a dedicated entity that is able to assign data to them, which is appropriate in a certain context. In addition, events from every GUI design unit of the hierarchy, which are significant for the further event handling or application data, have to be integrated in the presentation logic flow or event processing chain, so that individual units do not act isolated but create a comprehensive sequence of events.

Recent research [48][49] investigated on the role of task models for structuring the flow of dialogs. In analogy to the above described issue of division of labor for use case realizations between *ApplicationKernel* and *DialogKernel*, the architect has to decide on the responsibilities of a single *DialogKernel* concerning the flow of *Dialogs*. The question arises what part of the navigation is governed by higher situated components, e.g., a dedicated task controller, and what view changes are in the responsibility of the *DialogKernel*.

Large AT software portions. Thirdly, the Quasar software categories serve a main purpose to separate application from technical aspects, and thus, avoid *AT* software.

As far as the GUI architecture is concerned, we identified two aspects where *AT* software does regularly occur. The *Presentation* communicates with both the *GUI Framework* and *DialogKernel* in order to retrieve and store data inputs from the user. Eventually, the *Technical Data Models* of the *GUI Framework* and the *Dialog Data Model* have to be converted in the respective formats to enable information exchange. There may be a second conversion necessary between *Dialog Data Model* and *Domain Data Model* when the *DialogKernel* has to use a different data format.

Another aspect of *AT* software is the transformation of the *Dialog Data Model* to visual representations, which are constructed by the *Presentation*. Accordingly, the *Presentation* needs to possess knowledge of both the proper selection, arrangement of UI-Controls and the usage, creation of the latter via the specific *GUI Framework* facilities. Besides the first two issues, these two *AT* software aspects can additionally increase maintenance efforts. To solve the third issue, conventional architectures will not suffice and specific designs for additional decoupling have to be invented. An initial approach was formulated by Siedersleben and Denert in [29].

Granularity of GUI pattern design units. Another GUI design issue could be identified that is cross-cutting along the previously described three GUI design problems. It is concerned with the proper sizing of GUI design units, or with respect to common GUI patterns, MVC triads [10]. In detail, the main objective is keeping the event processing chain of the GUI perfectly matched with the functional requirement side of the value creation chain represented by business processes and corresponding use cases. Ultimately, these two mental models of event flows have to be kept in close synchronization to be able to firstly realize requirements properly and secondly apply changes to the GUI system efficiently when requirements are altered or added. Simple MVC or even greater HMVC [5] or MVP [6][7] *Controllers* are quickly overburdened in their scope in the attempt to trace functional requirements of the value creation chain, and so, keep track individual steps of application control flow.

The introduced GUI software categories (Figure 10) shed light on the granularity problem as they clearly distinguish greater and lesser components like *Dialog Logic*, *Presentation*, *View Definition* and *Presentation Event Handling*.

Originally, the MVC and its derivates were not designed to address such complex and hierarchical structures within information systems. Please remember that the classic MVC was built with the assumption in mind having this architecture applied as the global architectural style: there were no additional units of application or domain related design (generally *A* software descendants in terms of Quasar software categories) besides *Models*.

Nowadays, application and presentation logic as well as business processes do pose a difference to that rather simple *Model* design of the past. Therefore, *Controllers* face a different scope inside the value creation chain. To be able to separate concerns and keep a high cohesion, *Controllers* need to be assigned a proper level of responsibilities within the GUI software category tree. This in turn requires a corresponding sizing of triads or other pattern based GUI design units.

Identification of GUI design unit instances. Besides the granularity problem, there is an additional conflict whether to provide a custom identified structure of MVP or HMVC instances with better overview due to the reduced set of design units or to adopt an easy to identify hierarchical structure of classic MVC [31] with small fine-grained triads that follow a stereotype assignment approach of GUI design units (every UI-Control potentially serves as a triad connected to a global *Model* or a part of it). It has to be considered that the classic MVC approach can only be relied on as far as the *Presentation* is concerned. A *Dialog Logic* or *DialogKernel* unit of design and their responsibilities cannot be covered and have to be realized by custom solutions. According to the HMVC or MVP approach, the *Controllers* couple the different triads for communication and navigation purposes, so that the evolution or maintenance of both *Presentation* and *Dialog Logic* or *DialogKernel* units of design is closely coupled. Finally, this approach needs a further separation of concerns to resolve the issue. A perfect distribution of responsibilities will be difficult to achieve, since there are only certain triad members to accept the set of responsibilities symbolized by the principal software categories *View Definition, View Navigation, Presentation Event Handling, Dialog Data Model* and *Dialog Event Handling*. These need to be distributed among the triad members.

3) User Interface Patterns and Solution Approaches

Before we draw our conclusions, we briefly discuss how the incorporation of UIPs for the *Presentation* component may directly or indirectly resolve some of the identified GUI design issues.

AT software. At first, the mixture of application and technical aspects can directly be avoided by the integration of UIPs. In this context, UIPs promise the reuse of visual layout and related interaction. Thereby, the stereotype parts therein would be implemented once and encapsulated in the UIP units. Then the *Presentation* could be composed of these pattern units and would specify their contents via parameters. The UIP implementations would directly depend on the *GUI Framework* and no longer each *Presentation* unit. Therefore, fewer efforts would have to be spent on programming with *GUI Framework* facilities in the long run when UIPs could

be reused extensively. The development could be focused on the *DialogKernel* design issues instead.

Event differentiation by software categories. To integrate UIPs in the *Presentation*, the differentiated software categories for event processing will be of great value. The differentiation of events is a fundamental preparation for UIP integration as they prepare the better adaptability and even exchange of *Presentation* units. Responsibilities would be centered in the *DialogKernel* to raise the flexibility of UIPs.

We favor a solution that corresponds to the responsibilities of the software category tree and identifies *Controller* like design units accordingly. In detail, we think about moving away from the concrete representation of visual elements in each *View* of any triad. *Controllers* on different *A* software levels should be established along with abstract to more concretely defined *View* contents: *Controllers* based on the *Presentation* sub-tree of software categories can be closer coupled to a *View*, than *Controllers* of the *Dialog Logic* sub-tree. For instance, for a *Dialog Logic* level based *Controller* a visibility could be defined that describes an associated *View* to be controlled in state with only abstract elements like inputs, outputs, commands and navigation signals (compare abstract user interface, abstract interaction components of reference [50]), since this level of detail is completely sufficient for this type of *Controller*. In addition, this design keeps the opportunity to easily change the concrete details of the concrete *Views* lower in hierarchy. The higher situated *Controllers* do not depend on the concrete details; as long as the number of view states and in- as well as output events remain the same, details of views concerning layout may freely be changed. *View* states will be relevant for the *Dialog Logic,* but not their concrete visual appearance. The *Dialog Logic* is decoupled and kept independent from *Views* in turn.

In common MVC architectures, the *Controllers* are closely coupled to the *View* they are associated with. When the *Views* are altered or exchanged, the *Controllers* need to be also adapted or will not be reusable at all. For UIPs, these circumstances are not desirable; some *Controller* tasks need to be stable and reusable, so that at least the design units controlling the presentation logic states remain unaffected.

The above described approach to a solution is exactly what UIPs may need: *Controllers* cannot rely on knowledge about the *View's* concrete visual composition, instead a small interface is required that is both used for communication between *Dialogs* and UIPs as units on *Presentation* level and for the configuration or instantiation of UIPs. The UIP just required to provide the states, in- and outputs of data required by the *Dialog Logic* part. Anyway, these requirements have to be met by any other *Presentation*, which may be not UIP based, in order to comply with the underlying use case. Therefore, we suggest that an abstract representation of the *Presentation* from the *Dialog Logic's* point of view is sufficient and are confident that this approach will improve software architecture quality.

UIP impacts. To conclude, the software category tree displays the dependencies among the occurring GUI responsibilities. When UIPs are to be integrated in the GUI software architecture, an architect is able to assess the impacts UIPs may have on the established relationships. In particular, he can decide what interactions require a different design for coupling in order to enable the reusability and exchangeability of UIPs. A first description of such assessments was presented in reference [51], but this was based on an earlier revision of the software category tree.

VI. Conclusion and Future Work

The scope of this work is a study of the prevailing issues of GUI architecture design. A software category tree on the basis of Quasar was elaborated, which displays common responsibilities for GUI architectures and their dependencies. This display is independent of any platform, framework or architecture pattern. In contrast, available patterns can be be detailed or adapted on that basis. Eventually, the identified and described responsibilities can be re-structured in a GUI software architecture that may serve as a basis for a standardization of UIP integration. When no concern is mixed-up, reuse of UIPs is principally facilitated.

With the aid of the software categories, we have analyzed the common GUI MVC pattern and the Quasar client reference architecture. As result, we identified pattern specific and general issues of relevance for design decisions within GUI architecture development. The herein applied method with a decomposition of software categories and the tracing to an architecture model can be applied for other domains to assess the separation of concerns, cohesion and coupling.

Software categories and their relationship to patterns and design. One might ask what the difference is between the reviewed GUI architecture patterns with the presented tables of their responsibilities and the software category model, which nearly contains the same set of responsibilities.

Foremost, the software category model of course contains each responsibility of the patterns and is partly sourced from them. Nevertheless, the difference of capital importance is that the patterns already contain roles or design units with their fixed interfaces, dependencies and associations. These comprise the design as a structural and behavioral pattern unit and cannot be altered without changing the entire pattern concept.

On the contrary, the software categories model the responsibilities not from a fixed role perspective but from a point of view what topic, entities and operation types with their intended purpose are required for a certain responsibility. Hence, responsibilities in the software category tree are based on differentiated areas of knowledge and not on structural relationships in the first place. The advantage of the software categories is that they can be re-assigned to different designs, so that developers can be assured of completeness when each of the software categories can be traced to the resulting design. In that way, the same tasks the patterns serve are realized but different variations in design can be probed in a controllable manner. The patterns do not enable such a fine-grained decomposition of their responsibilities and allow no easy modifications without compromising the pattern' characteristic effects or forces.

Finally, the software categories do not only allow the allocation of responsibilities to designs; they are essentially

supplemented with rules [6] that are to be applied on the design of interfaces between interacting entities. This concept of rules shall ensure an improvement of coupling and a reduction of dependencies.

Future work. The findings of this work will influence our further research into the implementation options for UIPs. The Quasar client proved to be the most advanced architecture publicly available. On the basis of the identified issues of that architecture, we will have to develop dedicated solutions to prepare a suitable target architecture for UIPs. We need to further assess the architecture variants outlined in our previous work [30]. The software categories will help us to plan and evaluate possible solutions. Whatever architecture variant will be favored, it definitely needs a software architecture of high quality with well separated concerns to accept UIPs as additional and reusable artifacts. The solution must resolve the identified GUI design issues to allow the integration of UIPs as artifacts that enable a reduction of efforts for the adaptation of GUIs. Finally, UIPs shall not add additional dependencies, otherwise they would make GUI software systems even more difficult to maintain.

The established GUI software category tree will help us to integrate UIPs into the existing responsibility relationships and keep control about their influence. However, the software category tree needs to be approved in practical applications and possibly requires a revision.

ACKNOWLEDGMENT

We like to express our gratitude to the companies, close friends and family members that took part in and contributed valuable results to the survey mentioned in the introduction.

REFERENCES

[1] S. Wendler and D. Streitferdt, “A Software Category Model for Graphical User Interface Architectures,” The Ninth International Conference on Software Engineering Advances (ICSEA 14) IARIA, Oct. 2014, Xpert Publishing Services, pp. 123-133, ISBN: 978-1-61208-367-4.

[2] M. Haft, B. Humm, and J. Siedersleben, “The architect’s dilemma – will reference architectures help?,” First International Conference on the Quality of Software Architectures (QoSA 2005), Springer LNCS 3712, Sept. 2005, pp. 106-122.

[3] T. Reenskaug, “Thing-Model-View-Editor. An example from a planningsystem,” Xerox PARC technical note, 1979.05.12.

[4] S. Alpaev, “Applied MVC patterns. A pattern language,” The Computing Research Repository (CoRR), May 2006, http://arxiv.org/abs/cs/0605020, 2015.06.01.

[5] J. Cai, R. Kapila, and G. Pal, “HMVC: The layered pattern for developing strong client tiers,” JavaWorld Magazine, http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html (2000), 2015.06.01.

[6] M. Potel, “MVP: Model-View-Presenter. The taligent programming model for C++ and Java,” Taligent Inc., 1996, http://www.wildcrest.com/Potel/Portfolio/mvp.pdf, 2015.06.01.

[7] A. Bower and B. McGlashan, “Twisting the triad. The evolution of the dolphin smalltalk MVP application framework,” Tutorial Paper for European Smalltalk User Group (ESUP), 2000, Object Arts Ltd., 2000, http://www.object-arts.com/downloads/papers/twistingthetriad.pdf, 2015.06.01.

[8] J. Smith, “WPF Apps With The Model-View-ViewModel Design Pattern,” Microsoft Developer Magazine, 2009, Februrary, https://msdn.microsoft.com/en-us/magazine/dd419663.aspx, 2015.06.01.

[9] A. Ferrara, “Alternatives To MVC,”, http://blog.ircmaxell.com/2014/11/alternatives-to-mvc.html, 2015.06.01.

[10] A. Karagkasidis, “Developing GUI applications: architectural patterns revisited,” The Thirteenth Annual European Conference on Pattern Languages of Programming (EuroPLoP 2008), CEUR-WS.org, July 2008.

[11] M. Scarpino, SWT/JFace in action. Greenwich: Manning, 2005.

[12] T. Hatton, SWT: a Developer’s Notebook. Beijing: O'Reilly, 2004.

[13] R. Steyer, Google Web Toolkit: Ajax-Applikationen mit Java. Unterhaching: entwickler.press, 2007.

[14] M. Haft and B. Olleck, “Komponentenbasierte Client-Architektur [Component-based client architecture],” Informatik Spektrum, vol. 30, issue 3, June 2007, pp. 143-158, doi: 10.1007/s00287-007-0153-9.

[15] M. Fowler, “GUI Architecures,” 18.07.2006, http://martinfowler.com/eaaDev/uiArchs.html, 2015.06.01.

[16] J. Siedersleben, Moderne Softwarearchitektur [Modern software architecture], 1st ed. 2004, corrected reprint. Heidelberg: dpunkt, 2006.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software. Reading: Addison-Wesley, 1995.

[18] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. New York: John Wiley & Sons, 1996.

[19] M. Lindvall and K. Sandahl, “Practical implications of traceability,” Software - Practice and Experience (SPE), vol. 26, issue 10, Oct. 1996, pp. 1161-1180.

[20] P. Mäder, O. Gotel, and I. Philippow, “Getting back to basics: promoting the use of a traceability information model in practice,” The Fifth Workshop on Traceability in Emerging Forms of Software Engineering, IEEE, May 2009, pp. 21-25.

[21] J. Siedersleben, “An interfaced based architecture for business information systems,” The Third International Workshop on Software Architecture (ISAW '98), ACM, Nov. 1998, pp. 125-128.

[22] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, MA: Addison-Wesley, 2004.

[23] J. Engel, C. Herdin, and C. Märtin, “Exploiting HCI pattern collections for user interface generation,” The Fourth International Conferences on Pervasive Patterns and Applications (PATTERNS 12) IARIA, July 2012, Xpert Publishing Services, pp. 36-44, ISBN: 978-1-61208-221-9.

[24] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, “Tool support for an evolutionary design process using patterns,” Workshop on Multi-channel Adaptive Context-sensitive Systems (MAC 06), May 2006, pp. 71-80.

[25] J. Engel and C. Märtin, “PaMGIS: A framework for pattern-based modeling and generation of interactive systems,” The Thirteenth International Conference on Human-Computer Interaction (HCII 09), Part I, Springer LNCS 5610, July 2009, pp. 826-835.

[26] K. Breiner, G. Meixner, D. Rombach, M. Seissler, and D. Zühlke, “Efficient generation of ambient intelligent user interfaces,” The Fifteenth International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 11), Springer LNCS 6884, Sept. 2011, pp. 136-145.

[27] M. J. Mahemoff and L. J. Johnston, “Pattern languages for usability: an investigation of alternative approaches,” The Third Asian Pacific Computer and Human Interaction Conference (APCHI 98), IEEE Computer Society, July 1998, pp. 25-31.

[28] J. Borchers, "A pattern approach to interaction design," Conference on Designing Interactive Systems (DIS 00), ACM, August 2000, pp. 369-378.

[29] J. Siedersleben and E. Denert, "Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur [How to build information systems? Thoughts on a standard architecture]," Informatik Spektrum, vol. 23, issue 4, Aug. 2000, pp. 247-257, doi: 10.1007/s002870000110.

[30] S. Wendler, D. Ammon, T. Kikova, I. Philippow, and D. Streitferdt, "Theoretical and practical implications of user interface patterns applied for the development of graphical user interfaces," International Journal on Advances in Software, vol. 6, nr. 1 & 2, pp. 25-44, 2013, IARIA, ISSN: 1942-2628, http://www.iariajournals.org/software/.

[31] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk 80," Journal of Object Oriented Programming, vol. 1, August/September, 1988, pp. 26-49.

[32] M. Fowler, Patterns of Enterprise Application Architecture. New Jersey: Addison-Wesley Professional, 2003.

[33] D. Collins, Designing Object-Oriented User Interfaces. Redwood City, CA: Benjamin/Cummings Publ., 1995.

[34] E. Horn and T. Reinke, Softwarearchitektur und Softwarebauelemente [Software architecture and software construction elements]. München, Wien: Hanser, 2002.

[35] J. Dunkel and A. Holitschke, Softwarearchitektur für die Praxis [Software architecture for practice]. Berlin: Springer, 2003.

[36] D. Greer, "Interactive Application Architecture Patterns," http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/, 2015.06.01.

[37] S. Borini, "Understanding Model View Controller," http://forthescience.org/books/modelviewcontroller/00_introduction/00_preface.html, 2015.06.01.

[38] J. Siedersleben (ed.), "Quasar: Die sd&m Standardarchitektur [Quasar: The standard architecture of sd&m]. Part 2, 2. edn. sd&m Research: 2003.

[39] Open Qusasar Sourceforge project, http://sourceforge.net/projects/openquasar/, 2015.06.01.

[40] B. Humm, "Technische Open Source Komponenten implementieren die Referenzarchitektur Quasar [Technical Open Source Components implement the Reference Architecutre of Quasar]," in: ISOS 2004 - Informationsysteme mit Open Source, H. Eirund, H. Jasper, O. Zukunft, Eds. Proceedings GI-Workshop, Gesellschaft für Informatik, 2004, pp. 77-87.

[41] B. Humm , U. Schreier, and J. Siedersleben, "Model-Driven development – hot spots in business information systems," Proceedings of the First European conference on Model Driven Architecture: foundations and Applications, Springer LNCS 3748 , pp. 103-114.

[42] S. Wendler, D. Ammon, I. Philippow, and D. Streitferdt "A factor model capturing requirements for generative user interface patterns," The Fifth International Conferences on Pervasive Patterns and Applications (PATTERNS 13) IARIA, IARIA, May 27 - June 1 2013, Xpert Publishing Services, pp. 34-43, ISSN: 2308-3557.

[43] J. Vanderdonckt, "A MDA-compliant environment for developing user interfaces of information systems," The Seventeenth International Conference on Advanced Information Systems Engineering (CAiSE 2005), Springer LNCS 3520, June 2005, pp. 16-31.

[44] R. Stafford, "Service Layer," in [32].

[45] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models," Proceedings of The Sixth International Conference on Human-Computer Interaction, INTERACT 1997, IFIP Advances in Information and Communication Technology, Springer, 1997, pp.362-369.

[46] K. Sokolova, M. Lemercier, and L. Garcia, "Android passive MVC: a novel architecture model for the android application development," The Fifth International Conference on Pervasive Patterns and Applications (PATTERNS 2013), IARIA, May 27 - June 1 2013, pp 7-12.

[47] S. Wendler and D. Streitferdt, "An analysis of the generative user interface pattern structure," International Journal On Advances in Intelligent Systems, vol. 7, nr. 1 & 2, pp. 113-134, 2014, IARIA, ISSN: 1942-2679, http://www.iariajournals.org/intelligent_systems/index.html.

[48] F. Radeke and P. Forbrig, "Patterns in task-based modeling of user interfaces," The Sixth International Workshop on Task Models and Diagrams for Users Interface Design (TAMODIA 07), Springer LNCS 4849, Nov. 2007, pp. 184-197.

[49] V. Tran, M. Kolp, J. Vanderdonckt, and Y. Wautelet, "Using task and data models for user interface declarative generation," The Twelfth International Conference on Enterprise Information Systems (ICEIS 2010), vol. 5, HCI, SciTePress, June 2010, pp. 155-160.

[50] E. Mbaki, J. Vanderdonckt, J. Guerrero, and M. Winckler , "Multi-level Dialog Modeling in Highly Interactive Web Interfaces," The Seventh International Workshop on Web-Oriented Software Technologies (IWWOST 2008), ICWE 2008 Workshops, pp.38-43.

[51] S. Wendler and D. Streitferdt, "The Impact of User Interface Patterns on Software Architecture Quality," The Ninth International Conference on Software Engineering Advances (ICSEA 14) IARIA, Oct. 2014, Xpert Publishing Services, pp. 134-143, ISBN: 978-1-61208-367-4.

Model Inference and Automatic Testing of Mobile Applications

Sébastien Salva
LIMOS - UMR CNRS 6158
Auvergne University, France
email: sebastien.salva@udamail.fr

Patrice Laurençot
LIMOS - UMR CNRS 6158
Blaise Pascal University, France
email: laurenco@isima.fr

***Abstract*—We consider, in this paper, the problem of automatically testing Mobile applications while inferring formal models expressing their functional behaviours. We propose a framework called *MCrawlT*, which performs automatic testing through application interfaces and collects interface changes to incrementally infer models expressing the navigational paths and states of the applications under test. These models could be later used for comprehension aid or to carry out some tasks automatically, e.g., the test case generation. The main contributions of this paper can be summarised as follows: we introduce a flexible Mobile application model that allows the definition of state abstraction with regard to the application content. This definition also helps define state equivalence classes that segment the state space domain. Our approach supports different exploration strategies by applying the Ant Colony Optimisation technique. This feature offers the advantage to change the exploration strategy by another one as desired. The performances of *MCrawlT* in terms of code coverage, execution time, and bug detection are evaluated on 30 Android applications and compared to other tools found in the literature. The results show that *MCrawlT* achieves significantly better code coverage in a given time budget.**

***Keywords*-Model inference; Automatic testing; Android applications.**

I. Introduction

One of the primary purposes of software testing is to assess the quality of the features offered by an application in terms of conformance, security, performance, etc., to discover and fix its defects. Traditionally, testing is performed by means of test cases written by hands. But manual testing is often tedious and error-prone. Model-based Testing is another well-known approach, which automates the test case generation from a formal model describing the functional behaviours of the application. MbT makes possible the generation of exhaustive test suites (composed of all combinations of input values), but a complete model expressing all the expected behaviours of an application is then required. Unfortunately, writing complete models is often a long, difficult, and tedious task. As a consequence, only partial models are often proposed and available for testing. This makes MbT less interesting and even impractical with many real systems.

For specific applications, model inference methods based upon automatic testing can strongly help in the design of models. In particular, GUI applications (a.k.a. event-driven applications) belong to this category. Such applications offer a Graphical User Interface (GUI) to interact with and respond to a sequence of events played by the user. Partial models can be inferred by exploring (a.k.a. crawling) interfaces with automatic testing approaches. Furthermore, a large part of the application defects can be detected during the process. Afterwards, these generated models may be manually extended, analysed with verification techniques or employed for generating test cases.

This work falls under the automatic testing category and tackles the testing and the generation of functional models for Mobile applications. It provides additional details over [1] on various aspects, e.g., the use of strategies to explore applications. Several works already dealt with the crawling of GUI application, e.g., Desktop applications [2], Web applications [3], [4], [5] or Mobile ones [6], [7], [8], [9]. These approaches interact with applications in an attempt to either detect bugs or record models or both. These previous works already propose interesting features, such as the test case generation from the inferred models. However, it also emerges that many interesting issues still remain open. Firstly, performing experiments with the GUIs of Web or Mobile applications may lead to a large and potentially unlimited number of states that cannot be all explored. Additionally, the application traversing is usually guided by one of these strategies: DFS (Depth First path Search) or BFS (Breadth First path Search). These are relevant on condition that all the application states would be explored. But when the application state number is large or the execution time is limited, using other strategies could help target the most interesting application features as a first step.

This paper contributes in these issues by proposing a framework called *MCrawlT*. Its goals are to experiment Mobile applications to infer both storyboards and formal models, and to detect bugs. It also aims to achieve good code coverage quickly. The originality of our approach lies in the following features:

- model definition and state abstraction: we use PLTSs (Parametrised Labelled Transition Systems) as models that we specialise to capture the functional behaviours of Mobile applications. Most of the model inference approaches use state abstractions to produce models. But they often either face the problem of state space

explosion or produce too abstract models that do not capture sufficient information to later perform analysis and testing. Both issues often occur because of an inappropriate and unmodifiable state abstraction definition. Here, we propose a flexible PLTS state representation which allows the definition of state abstraction with regard to the application content. This PLTS state definition also helps define state equivalence classes, which slice a potentially infinite state space domain into finite equivalence classes. Our algorithm aims at exploring every discovered state equivalence classes once. As a consequence, our algorithm terminates;

- exploration strategies performed in parallel: we propose a first algorithm which uses exploration strategies to target specific parts of a Mobile application. The algorithm is based upon the Ant Colony Optimisation (ACO) technique and simulates several ants represented by threads, which explore application states and lay down pheromones. These pheromone trails, built in parallel, allow the ants to target the most relevant states w.r.t. a chosen strategy. A strategy can be replaced by another one as desired;
- code coverage enhancement: GUI application testing approaches traditionally start exercising an application from its root interface. Nevertheless, we observed that some application features cannot be automatically tested and hence block the application exploration, which may lead to low code coverage. This is why we propose an extended algorithm which tries to cover an application starting from each of its available interfaces and which infers several PLTSs along the execution. If a blocking feature is bypassed, the application is deeper covered and therefore the code coverage is improved. Furthermore, the algorithm avoids exploring states previously encountered to not build several identical models.

In collaboration with the Openium company, which is specialised in the design of Mobile applications, we have implemented *MCrawlT* for Android applications. The tool is publicly available at https://github.com/statops/MCrawlerT. We applied *MCrawlT* to 30 real-world Android applications and compared its effectiveness against other available automatic testing tool in terms of code coverage, execution time and bug detection.

The paper is structured as follows: Section II surveys related work and introduces our motivations. For expository purposes, we start by presenting an overview of our approach in Section III that we apply on the Ebay Mobile application, taken as example throughout the paper. We also give the assumptions that guided the design of our approach. Section IV gives definitions and notations about the PLTS model. In particular, we specialise the PLTS formalism for Mobile applications and give a state equivalence relation. Our exploration algorithms are detailed in Section V. Section VI presents experimental results. Conclusions are drawn in Section VII along with directions for further research and improvements.

II. Related Work and discussion

Several papers dealing with automatic testing and model inference approaches were issued in the last decade. Here, we present some of them relative to our work.

Several works were proposed for white-box systems. For example, *Contest* [6] is a testing framework which exercises smartphone applications with the generation of input events. This approach relies upon a systematic test generation technique, a.k.a. concolic testing, to explore symbolic execution paths of the application. Artzi et al. [4] proposed an automatic white-box testing approach for finding faults in PHP Web applications. The application code is covered using combined concrete and symbolic (concolic) execution, and constraint solving to detect execution failures and malformed HTML code. These white-box based approaches should theoretically offer a better code coverage than the automatic testing of black-box systems. However, the number of paths being explored concretely limits to short paths only. Furthermore, the constraints have not to be too complex for being solved. As a consequence, the code coverage of these approaches is not high in practice.

On the other hand, many black-box based methods were also proposed. Memon et al. [2] initially presented *GUI Ripper*, a tool for scanning and testing desktop applications. This tool produces event flow graphs and trees showing the GUI execution behaviours. Only the click event can be applied, and *GUI Ripper* produces many false event sequences which may need to be weeded out later. Furthermore, the actions provided in the generated models are quite simple (no parameters). This approach was extended to support Mobile applications in [10] with the tool *Guitar*. This one is based upon *GUI Ripper* but also supports the inference of Event flow graphs and test case generation. Mesbah et al. [3] proposed the tool *Crawljax* specialised in Ajax applications. It produces state machine models which capture the changes of the DOM structures of HTML documents by means of events (click, mouseover, etc.). An interesting feature of *Crawljax* is the concatenation of identical states in the model under construction. If two states, which represent the DOM structures of HTML documents, are similar, they are assembled together. This helps reduce the number of states which may be as large as the DOM modifications. In practice, to limit the state space and to avoid the state explosion problem, state abstractions have to be given manually to extract a model with a manageable size. *Webmate* [5] is another automatic testing tool for Web applications. It produces graphs showing the observed GUI and events.

Since our approach targets Mobile applications, we explore more cautiously this field in the following. *Monkey* [11] is a random testing tool proposed by Google. It is considered as a reference in many papers dealing with Android application automatic testing. However, it cannot simulate complex workloads such as authentication, hence, it offers light code coverage in such situations. The tool Dynodroid [9] exercises Android applications with UI events like *Monkey* but also with system events to improve code coverage. A similar technique is applied on Android applications in [12]. But, the approach additionally performs static analyses on Android application source codes to later guide the application exploration. No model is provided with these approaches. Amalfitano et al. [7], [13] proposed *AndroidRipper*, a crawler for crash testing and for regression test case generation. A simple model, called GUI tree, depicts the observed screens. Then, paths of the tree not terminated by a crash detection, are used to re-generate regression test cases. Joorabch et al. [14] proposed another crawler, similar to *AndroidRipper*, dedicated to iOS applications. Yang et al. proposed the tool *Orbit* [8] whose novelty lies in the static analysis of Android application source code to infer the events that can be applied on screens. Then, a classical crawling technique is employed to derive a graph labelled by events. This grey-box testing approach should cover an application quicker than our proposal since the events to trigger are listed by the static analysis. But *Orbit* can be applied only when source code is available. This is not the case for many Android applications though. The algorithm implemented in *SwiftHand* [15] is based on the learning algorithm L^* [16] to generate approximate models. The algorithm is composed of a testing engine which executes applications to check if event sequences meet the model under generation until a counterexample is found. An active learning algorithm repeatedly asks the testing engine observation sequences to infer and potentially regenerate the model w.r.t. all the event and observation sequences.

We deduced from these papers the main following key observations:

1) to prevent from a state space explosion, the approaches that infer models, e.g., [2], [8], [17], usually represent application states in a fixed manner and with a high level of abstraction. This choice is particularly suitable for comprehension aid, but these models often lack information for test case generation. In contrast, other approaches try to limit the model size on the fly. The algorithms introduced in [3], [13] concatenate identical states of the model under construction, but the resulting model does not capture the same behaviours as those expressed in the original model. Such an extrapolated model may lead to false positives if used for test case generation. We propose here another solution based upon the PLTS formalism and the definition of state equivalence classes. We specialise the PLTS for Mobile applications to ease the definition of state abstraction. Users can modify the latter to build models as desired. We define state equivalence classes to segment the potentially infinite state space domain of an application in a finite manner. As a consequence, we show that our algorithm terminates. Finally, we use a bisimulation minimisation technique [18] to reduce the PLTS size. This technique offers the advantage to preserve behavioural equivalence between models;
2) many inference model methods consist in analysing and completing interfaces with random test data and triggering events to discover new interfaces that are recursively explored in an in-depth manner. As a consequence, the application exploration is usually guided with a DFS strategy. When an application returns a high number of new interfaces, the graph to be explored may become too large to visit in a reasonable time delay. The search is only performed to a limited depth, and the explored section of the application is not necessarily the most interesting one. We believe that a strategy choice is relevant when the execution time is limited, for instance or when an insight of the application functioning (code structure) is known or both. Indeed, strategies allow to quicker target some application features. Our algorithm is based upon the ACO technique in order to accept a large strategy set. For instance, our algorithm supports semantics-based strategies, i.e., strategies guided by the content found in applications screens. Furthermore, the ACO technique is known as a good heuristic to cover paths through graphs in parallel;
3) crash reporting is another feature supported by some of these methods. Stress testing is performed in [11], [19], [9] for trying to reveal more bugs, for instance by using random sequences of events. Besides, the tool *AndroidRipper* [7] generates test cases for each observed crash. Our approach also performs stress testing: like Monkey [11], random sequences of events are applied on screens. We also use well-known values for revealing bugs for testing. Our tool reports the observed bugs and generates one test case for each as well. These ease the analysis of the detected errors and help deduce whether some errors are false positives. As crash reporting and detection are not original features, we do not detail this part in the paper. We only discuss about crash detection in the section dealing with the evaluation of our framework (Section VI).

We presented in [1] a rudimentary introduction of this work describing an initial algorithm based upon the ACO technique. In this paper, we define another model, state equivalence classes, and we revisit the exploration algorithm to better match the concept of the ACO technique. Then,

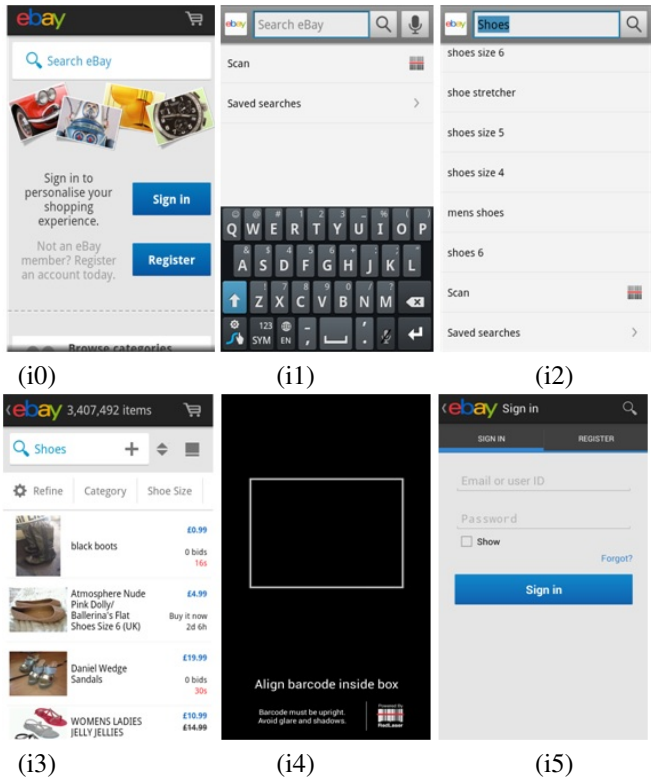


Figure 1: Ebay Mobile Screen examples

we propose a second Exploration algorithm to enhance code coverage and we show that our algorithms terminate. Finally, our evaluation focuses on much more applications and criteria.

III. Overview

In this section, we introduce the terminology used throughout the paper and a motivating example on which we apply our framework *MCrawlT*. The formal details and *MCrawlT* algorithms are introduced in Section V.

A. Terminology and assumptions

Generally speaking, we say that users expect screens to interact with Mobile applications. We consider that a screen represents one application state, the number of states being potentially infinite. A screen is built by a GUI application component, e.g., a class. We call them *Activities* (in reference to Android applications). These components display screens by instantiating *Widgets* (buttons, text fields, etc.) which are often organised into a tree structure. They also declare the available (UI) events that may be triggered by users (click, swipe, etc.). A Widget is characterised by a set of properties (colour, text values, etc.). Hence, one Activity can depicts several screens, composed of different Widgets or composed of the same Widgets but having different properties.

Figure 1 depicts some screen examples of the *Ebay Mobile* application, which is available on the *Google Play* store (https://play.google.com/store). This complex application includes 135 Activities and we only depict five of them in Figure 1. The initial screen is loaded by the Activity *eBay* (i0). A user may choose to search for an item by clicking on the editable text field Widget. In this case, the Activity *RefineSearch* is reached (i1). For instance, if the user enters the keyword "shoes", the search result list is displayed in the screen i2; the Activity *RefineSearch* is unchanged but its content (Widgets) is. Then, three new Activities may be reached: 1) an Activity called *SegmentSearchResult* (i3) displays a result when one element of the proposed list in i2 is chosen, 2) a *Scanner* Activity is started when the text field "Scan" is clicked (i4) and 3) a log-in process is performed when the "saved searches" item is selected (Activity *SignIn*, i5). Now if we replace the value "shoes" by any other String value, one can easily deduce that this application can yield a huge state number.

B. Assumptions

The purpose of our algorithm is to generate input events in order to feed a Mobile application with respect to an exploration strategy to achieve formal models and good code coverage quickly. To design this approach, we had to assert the following hypotheses:

Mobile application testing: we consider black box applications which can be exercised through screens. It is possible to dynamically inspect the states of the running application (to collect Widget properties). This assumption holds with many recent GUI applications (Web applications, Mobile applications, etc.). The set of user events enabled on a screen should be collected as well. If not, Widgets provide enough information (type, etc.) to determine the set of events that may be triggered. Otherwise, our algorithm considers them all for testing an application. Furthermore, any new screen can be observed and inspected (including application crashes),

Application reset: we assume that Mobile applications and their environments (database, Operating System) can be reset,

Back mechanism availability: several operating systems or applications (Web navigators, etc.) also propose a specialised mechanism, called the *back mechanism* to let users going back to the previous state of an application by undoing the last event. We do not consider that this mechanism is necessarily available and, if available, we assume that it does not always allow to go back to the previous state of an application (modified implementation, unreachable state, etc.). Most of the other methods assume that the back mechanism always works as expected, but this is frequently not the case.

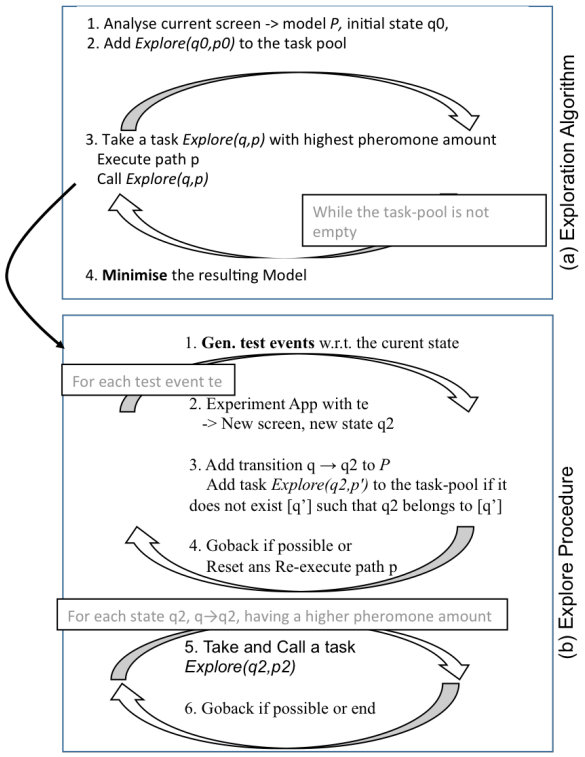


Figure 2: Algorithm overview

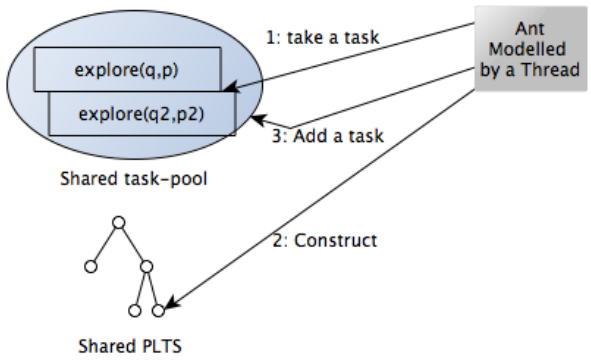


Figure 3: Parallel exploration functioning

C. Exploration algorithm overview

An overview of our algorithm is depicted in Figure 2. It is framed upon the Ant Colony Optimisation (ACO) technique to support the definition and the use of exploration strategies, which can be applied with concurrent threads. With the ACO technique, the optimal path search in a graph is performed by simulating the behaviour of ants seeking a path between their colony and a source of food: firstly, ants explore randomly and lay down little by little pheromone trails that are finally followed by other ants.

Ants are here modelled with threads that explore application states having the highest pheromone amount, earlier put down by other ants. This part is implemented, as described in Figures 2(a) and 3 by using the task-pool paradigm associated with tasks of the form $Explore(q, p)$ with q the state to visit and p the path allowing to reach q from the initial state $q0$ of the application. Intuitively, this path corresponds to a trail previously constructed by ants. Initially, the first screen of the application under test is modelled with an initial PLTS state q_0. The exploration of a screen, modelled with a PLTS state q, is conveyed with a task $Explore(q, p)$, which is placed into the task-pool, implemented as an ordered list in descending order. A thread picks out a task having the highest pheromone amount, reaches the state q and starts the exploration. Once the task-pool is empty, the application exploration is over and a PLTS $\mathcal{P}$ is achieved. This one is minimised to reduce the PLTS state set.

The execution of a task $Explore(q, p)$ is achieved by the *Explore* procedure, illustrated in Figure 2(b)). The latter consists in generating a set of test events (parameter values combined with an event set) w.r.t. the current application state. Each test event is applied on the application to reach a new screen. This one is interpreted and modelled as a new state q_2. However, this step may lead to infinite state space domains and endless explorations. To avoid this issue, the algorithm slices state space domains into finite equivalence class sets by means of a relation defined upon the state content (see Section IV). We have chosen to explore one state per equivalence class to keep a reasonable model size but this could be modified. The algorithm completes the model with a transition $q \xrightarrow{q_2} 2$ and finally tries to backtrack the application to go back to its previous state by undoing the previous action. If the back mechanism cannot be triggered, the application is restarted from its initial state $q0$. Once the state q is explored, the thread can explore a next state iff it includes a pheromone amount higher than the one found in q. Likewise, if the back mechanism cannot be applied, the *Explore* procedure execution terminates here. The thread continues its execution in Algorithm 1 by taking another task in the task-pool.

Figure 4 illustrates how our algorithm works on the *Ebay Mobile* application. We have chosen the DFS exploration strategy whose implementation is detailed in Section V-B. With this strategy, the deeper a state is in the model from the initial state, the higher the pheromone amount is. In order to show a comprehensive but yet concise illustration, we use only two text field values "shoes" and "All shoes" to fill the editable text fields found in screens. Furthermore, we consider that the back mechanism is available and that the state equivalence relation is: *two states are similar if they have the same Widget properties except those related to the text field values*. These last Widget properties are usually not considered for conceiving state abstractions since these often lead to a large and potentially infinite set of states.

1) initially, the first screen of the application (Figure 1(i0)) is inspected to derive a first task

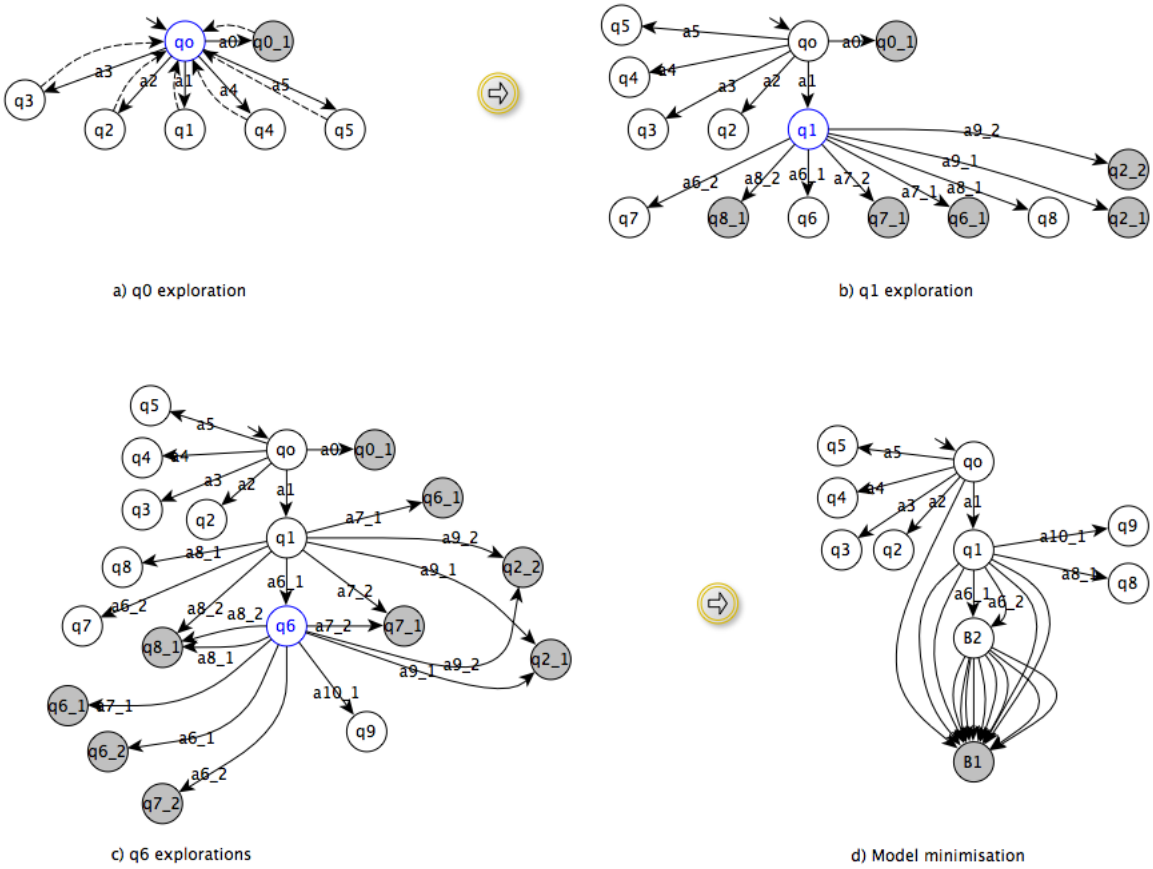


Figure 4: Model inference progress on the Ebay application

$Explore(q0, p0 = \emptyset)$. $q0$ is derived from the Widget properties extracted from the screen. It also includes a pheromone amount equal to 0. $[q0]$ is the initial state equivalence class. Then, the task $Explore(q0, p0 = \emptyset)$ is chosen by the algorithm. The outcome of this task is depicted in Figure 4(a). A list of test events that can be applied on the current screen is constructed: intuitively, these events aims at clicking on the 2 buttons, the 2 images, or the text field *home_search_text* found in the current screen. Some events are detailed in Figure 5. When the test event $a0$ (click on the Widget id/home) is executed, a new screen is observed. The Widget properties are extracted to construct the state $q0\_1$. This state is marked as final (in grey in the figure) since it has the same Widget properties as $q0$, except for the text field values. In other words, $q0\_1$ belongs to the state equivalence class $[q0]$. The transition $q0 \xrightarrow{a0} q0\_1$ is added to the model. Then, we call the back mechanism to go back to $q0$. This event is illustrated with dashed transitions in Figure 4(a). For the other test events, every time a new screen is found, a new state and a new transition are added to the model. These states include a pheromone amount increased by one unit to meet the DFS strategy. For each state, a new equivalence class is created, and a new task is also put into the task-pool;

2) the algorithm now takes the next task having a state with the highest pheromone amount. In this case, the task $Explore(q1, q0 \xrightarrow{clickhome\_search\_text} q1)$ is picked out. $q1$ represents the "RefineSearchAct" Activity (Figure 1(i1)). This task gives the PLTS of Figure 4(b) but to keep this figure readable, we intentionally do not add the transitions which express the calls of the back mechanism. As before, a list of test events is generated. $q1$ is experimented with these and new states, e.g., $q6, q7, q8$, are observed. For instance, $q6$ is obtained by clicking on the Widget *up* and by filling the editable Widget *search_text* with the value "shoes". $q7$ is reached in the same way but by using the value "All shoes". The two states $q6$ and $q7$ are obtained from the Activity "RefineSearch" but they differ from each other on the Widget *listview* which is a container. In $q6$, *listview* has more items than in $q7$, and consequently, they are not similar and do not belong to the same equivalence class. The events $a8\_1$ or $a8\_2$ (clik:widget=id/text2) lead

to the Scanner Activity (Figure 1 (i4)). The event $a8\_1$ is firstly executed. We observe a new state $q8$, which has to be explored. The event $a8\_2$ leads to the same screen from which a state $q8\_1$ is derived. $q8\_1$ exactly includes the same Widget properties as $q8$ but it belongs to the equivalence class $[q8]$ and is then marked as final. The new states discovered during this step include a pheromone amount increased by one unit;

3) the task $Explore(q6, q0 \xrightarrow{clickhome\_search\_text} q1 \xrightarrow{clickup,search\_text="shoes"} q6)$ is now chosen in the task-pool. This state expresses the Activity of Figure 1(i3). As in the previous steps, test events are generated to experiment the current screen. For sake of readability, we chose to only consider the event a_10 with the container "listview", which stands for "the click on the first item of listview". When the event a_10 is triggered, a new state $q9$ is found and another equivalence class is created. In contrast, for the other events, all the arrival states belong to an existing equivalent class and are marked as final. We obtain the model illustrated in Figure 4(c);
4) even though the algorithm should continue with the task composed of the state $q9$, we assume here that the task-pool is empty to keep the example concise. The model of Figure 4(c) is finally minimised. All the final states are merged to one unique state B1 as illustrated in Figure 4(d). States $q6$ and $q7$ are aggregated into $B2$ since the same behaviours can be observed from both states. The minimisation process is detailed in Section V.

In this example, we have shown that the algorithm discovers trails into the applications by laying down pheromone amounts. The generated models contain the events and screens observed while testing. As described previously, this algorithm works with one thread. But, the task-pool paradigm is particularly suitable to run a group of threads exploring states in parallel.

In the following, we describe the functionalities, the model and equivalence relation definitions, succinctly suggested previously. Algorithm 1, given in Section V, implements with details this overview.

IV. Mobile application modelling

In this section, we introduce a few definitions and notations to be used throughout the paper.

We use PLTS (Parameterised Labelled Transition System) as models that we specialise to represent Mobile application behaviours. The PLTS is a kind of automata model extended with variable sets. The use of variables helps describe valued actions composed of parameter values and to encode the states of a system.

Before giving the model definition, we give some notations on variables. We assume that there exist a domain of values denoted D and a variable set X taking values in D. The assignment of variables in $Y \subseteq X$ to elements of D is denoted with a mapping $\alpha : Y \rightarrow D$. We denote D_Y the assignment set over Y. Given two assignments $\alpha_1 \in D_Y$ and $\alpha_2 \in D_Z$ with $Y \cap Z = \emptyset$, their union is defined as $\alpha_1 \cup \alpha_2(x) = \alpha_1(x)$ iff $x \in Y, \alpha_2(x)$ iff $x \in Z$. An example of assignment is $\alpha = \{x := "blue", y := 1\} \in D_{x,y}$.

Definition 1 (PLTS) *A PLTS (Parameterised Labelled Transition System) is a tuple $< V, I, Q, q0, \Sigma, \rightarrow >$ where:*

- *$V \subseteq X$ is the finite set of variables, $I \subseteq X$ is the finite set of parameters used with actions,*
- *Q is the finite set of states, such that a state $q \in Q$ is an assignment over D_V,*
- *q_0 is the initial state,*
- *Σ is the finite set of valued actions $a(\alpha)$ with $\alpha \subseteq D_I$,*
- *$\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation. A transition $(q, a(\alpha), q')$ is also denoted $q \xrightarrow{a(\alpha)} q'$.*

Below, we adapt this generalised PLTS definition to express Mobile application properties, i.e., screens and events.

UI event representation: We interact with Mobile applications by means of events, e.g., a click, applied on Widgets. Furthermore, editable Widgets are possibly completed before triggering events. We capture these events with PLTS actions of the form $event(\alpha)$ with $\alpha = \{widget := w, w_1 := val_1, ..., w_n := val_n\}$ an assignment over D_I; the parameter $widget$ denotes the Widget name on which the event is applied, and the remaining variables are assignments on Widget properties. We also denote the triggering of the back mechanism with the action $back(\alpha)$ with α an empty assignment.

Mobile application state representation: We concluded from the literature that some Widget properties are considered as more important than others to encode Mobile application states. These properties usually indicate a strong application behaviour modification and take only a few values to prevent from a state space explosion. We denote WP the set of these Widget properties. It often gathers properties related to the Widget visibility, size, position, colour, etc. The properties that usually take a lot of different values, e.g., the properties about text field values, are not chosen to identify Mobile application states. Consequently, in the remainder of the paper, we consider that WP is composed of all Widget properties except those related to text field values.

We specialise PLTS states to store the content of screens (Widget properties) in such a way as to later facilitate the construction of state equivalence classes. We define a PLTS state q as a specific assignment of the form $act \cup wp \cup wo \cup end \cup ph$ where:

- act is an assignment returning an Activity name,
- (wp, wo) are two sets of Widget property assignments. The union of wp and wo gives all the Widget property

values found in an application screen displayed by act. We keep in wp the Widget properties of WP that indicate a strong application behaviour modification and that take only a few values. The other property assignments are placed into wo,

- end is a boolean assignment marking a state as final,
- ph is an assignment related to the exploration strategy, which stores a pheromone amount.

For readability, a state $q = act \cup wp \cup wo \cup end \cup ph$ is denoted (act, wp, wo, end, ph).

This state structure greatly eases the definition of the state equivalence relation given below. This one shall be particularly useful to determine if a state belongs to an existing equivalent class and requires to be explored or not.

Definition 2 (State equivalence relation) *Let $\mathcal{P} =< V, I, Q, q0, \Sigma, \rightarrow >$ be a PLTS and for $i = 1, 2$ let $q_i = (act_i, wp_i, wo_i, end_i, ph_i)$, be a state in Q. We say that q_1 is equivalent to q_2, denoted $q_1 \sim q_2$ iff $act_1 = act_2$ and $wp_1 = wp_2$. $[q]$ denotes the equivalence class of equivalent states of q. $Q/\sim$ is the set of equivalence classes in Q.*

This definition, combined with our algorithm, gives a very adaptable state equivalence relation which can be modified according to the WP set. As stated previously, we consider that WP is initially composed of all Widget properties except those related to text field values. But if, for an application, a Widget property takes a large number of values in WP, this one can be removed from WP to obtain a constricted set of equivalence classes and to achieve a finite exploration.

Let us consider an application including advertising strips that are continuously updated. We assume that the Widget property related to the advertising display is denoted $w.content$. This property takes a potentially infinite number of values and may lead to the state space explosion problem while generating the PLTS. Indeed, for an Activity act which holds $w.content$, the exploration algorithm shall reach several states $q_i = (act, wp_i, wo_i, end_i, ph_i)_{(i>1)}$ that are almost similar except that they contain in wp_i different assignments $w.content := val_i$ related to the different advertisings. Each state q_i involves a new equivalence class $[q_i]_{(i>1)}$. The application exploration will likely not terminate. The removal of the property $w.content$ in WP fixes this problem. In this case, the algorithm now constructs different states $q_i = (act, wp, wo_i, end_i, ph_i)_{(i>1)}$ such that the different assignments $w.content := val_i$ are now placed into wo_i. But, the algorithm builds one equivalence class $[q]$ since the assignment wp is unchanged. Therefore, when the algorithm reaches a state $q_2 = (act, wp, wo_2, end_2, ph_2)$, q_2 belongs to $[q]$ and is thus marked as final. Only one state of $[q]$ is explored, hence, the exploration is finite.

This example also shows that if WP is only composed of discrete variables taking a finite number of values, then the number of state equivalence classes in a PLTS is bounded. This is captured by the following Proposition, which shall be particularly useful to prove the termination of our algorithms.

Label	Action
$a0$	click(widget:=id/home)
$a1$	click(widget:=id/home_search_text)
$a2$	click(widget:=id/button_sign_in)
$a3$	click(widget:=id/button_register)
$a4$	click(widget:=id/rtmImageView)
$a5$	click(widget:=id/home_settings)
$a6\_1$	click(widget:=id/up, search_src_text:=”All shoes”)
$a6\_2$	click(widget:=id/up, search_src_text:=”shoes”)
$a7\_1$	click(widget:=id/search_button, search_src_text:=”All shoes”)
$a7\_2$	click(widget:=id/search_button, search_src_text:=”shoes”)
$a8\_1$	click(widget:=id/text1, search_src_text:=”All shoes”)
$a8\_2$	click(widget:=id/text1, search_src_text:=”shoes”
$a9\_1$	click(widget=id/text2, search_src_text:=”All shoes”)
$a9\_2$	click(widget:=id/text2, search_src_text=”shoes”)
$a10\_1$	click(widget:=”listview at position 1”, search_src_text:=”shoes”)

Figure 5: Actions and Guards of the PLTS

Proposition 3 *Let $\mathcal{P} =< V, I, Q, q0, \Sigma, \rightarrow >$ be a PLTS modelling a Mobile application App. Let n be the number of Widget properties of WP. If m is the maximum number of values that any Widget property can take during the testing of App, then $card(Q/\sim) \leq m^n$.*

Sketch of proof: all the screens of App are expressed with states of the form $q = (act, wp, wo, end, ph)$. At most, we have m^n different assignments wp. Two states q_1, q_2 are equivalent iff $act_1 = act_2$ and $wp_1 = wp_2$ (Definition 2). But if there is no equivalent state, we have at most m^n equivalence classes including one state.

The choice of the Widget properties to keep in WP is left to users. Intuitively, the more a Widget property of WP takes values, the larger the generated models is. We observed that ignoring the Widget properties related to text field values is mostly sufficient. But sometimes, the first exploration of an application makes emerge some properties that need to be removed to achieve a finite model in a reasonable time delay.

Figures 4(c), 5, and 6 illustrate a PLTS example derived from the Ebay Mobile application, after covering only 5% of its Activities. We detail some PLTS states in a reduced form in Figure 6: we give the Activity name, the numbers of Widget properties (wp and wo), and the assignments end and ph. The PLTS actions are given in Figure 5. For instance, the state $q0$ represents the initial Activity *eBay* of the application, which includes 2 buttons, 6 images, and 7 text fields . $q1$ is reached from $q0$ by executing the action $a1$, i.e., by clicking on the $home\_search\_text$ Widget.

V. Automatic testing and model inference with ACO

In this section, we formally describe the algorithms implemented in *MCrawlT*. We firstly detail the algorithm

State	Activity	#wp	#wt	end	ph
$q0$	eBay	2b,2im	6t,1e	false	0
$q1$	RefineSearchAct	2b,4im	3t,1e	false	1
$q2$	SignAct	2b,4im	2t,1e	false	1
$q6$	RefineSearchAct	2b,3im,40l_e	3t,1e	false	2
$q6\_1$	RefineSearchAct	2b,3im,40l_e	3t,1e	true	2
$q7$	RefineSearchAct	2b,3im,105l_e	3t,1e	false	2
$q7\_1$	RefineSearchAct	2b,3im,105l_e	3t,1e	true	2

b: button e: editable text field t: text field im: image
l_e: # elements in the listview Widget

Figure 6: Summary of some states of the PLTS

considered in the overview. Then, we propose an extended version, which aims at improving code coverage. We also provide (time) complexity results.

A. Model inference Algorithm 1

Our solution is framed upon the PLTS formalism to infer formal models. The combination of the PLTS state definition with the state equivalence relation segments the potentially infinite state space domain into a finite set of equivalence classes and every class is visited once. Our algorithm is also based upon the ACO technique to perform explorations in parallel and to support different application strategies. Algorithm 1 implements the initial part of this solution. The ACO technique is implemented with the task-pool paradigm where the tasks of the pool are executed in parallel on condition that the tasks are independent. This is the case here since several application instances can be experimented into independent test environments (smartphones or emulators). All the threads share the same PLTS $\mathcal{P}$ and the same task-pool implemented as an ordered list in descending order. For sake of readability, we assume that these shared resources are protected against concurrent accesses.

Algorithm 1 takes a Mobile application App as input and launches it to analyse its first screen and to initialise the first state $q0 = (act, wp, wo, end := false, ph_0)$ of the PLTS $\mathcal{P}$. q_0 is obviously not marked as final and includes a pheromone amount related to the chosen strategy. This initial step is carried out by one thread only. Afterwards, the interface exploration begins: each available thread executes the loop of Algorithm 1 (line 7): it picks out a task $Explore(q, p)$, which corresponds to the exploration of the state q, such that q holds the highest pheromone amount. Before exploring q, an instance of the application is launched in a re-initialised test environment and q is reached from q_0 by covering and executing the actions of the PLTS path p. Once there is no more task to perform, a second PLTS $\mathcal{MP}$ is computed with a minimisation technique. This PLTS minimisation aims to yield more compact and readable models for comprehension aid.

One task, pulled from the task-pool, is now executed by calling the *Explore* procedure, which somehow simulates an ant exploring a state and laying down pheromones. Initially, we added a stopping condition limiting the execution time. This condition was used in the experiments presented in Section VI. The *GenEvents* procedure is called and generates test events used to feed the application. It starts by analysing the current screen, extracts the editable Widgets, and produces a set of assignments expressing how completing these editable Widgets with values. Similarly, the events that can be triggered on the Widgets are dynamically detected. We obtain a set $Events$ composed of $event(\alpha)$ with α an assignment of the form $\{widget := w, w_1 := val_1, ..., w_n := val_n\}$. Then, the exploration of the current state q begins (line 6). The editable Widgets are completed, and an event is triggered with respect to the test event $event(\alpha)$. It results in a new screen $Inew$ (line 7), which is analysed to extract the assignments constituting the state q_2. The pheromone amount, which is laid down in q_2, is computed with the *Ph_Deposit* procedure. This one implements the exploration strategies. The algorithm now checks whether this new screen and its corresponding state q_2 have to be explored. Naturally, if $Inew$ reflects the termination of the application (exception, crash), q_2 must not be explored. As stated previously, we have chosen to explore one state for each equivalence class. Hence, if q_2 belongs to a previously discovered equivalence class $[q']$ in $Q/\sim$ then q_2 is marked as final with the assignment $end := true$ and is not explored. Otherwise, q_2 has to be explored and a new task $Explore(q_2, p.t)$ is added to the task-pool (lines 14-16). In both cases, a new transition carrying $event(\alpha)$ and leading to q_2 is added to the PLTS $\mathcal{P}$. To apply the next input event $event(\alpha)$, the application has to go back to its previous state by undoing the previous interaction. This is done with the *Backtrack* procedure (line 17) whose role is to undo the most recent action. When the direct interface restoration is not possible, the *Backtrack* procedure returns false and

Algorithm 1: Mobile application exploration V1

input : Application App
output: PLTS $\mathcal{P}$, $\mathcal{MP}$

```
// Initialisation performed by one thread only
1  Start the application App;
2  Analyse the current screen → Activity act, the Widget property
   lists wp, wt;
3  Initialise ph_0 (depends on the chosen strategy);
4  Initialise PLTS P with q0 = (act, wp, wo, end := false, ph_0);
5  Q/~ = {[q0]};
6  Add (Explore(q0, p = ∅)) to the task-pool;
   // code executed in parallel, P, task-pool,
      Q/~ are shared
7  while the task pool is not empty do
8      Take a task (Explore(q, p)) such that
       q = (act, wp, wt, end, ph) includes the highest pheromone
       amount ph;
9      Reset and Execute app by covering the sequence of actions
       of p;
10     Call Explore(q,p);
   // code executed by one thread
11 MP:= Minimise(P);
```

Procedure Explore

1 Procedure $Explore(q, p)$;
2 **if** *[processing time > T]* **then**
3 stop;
4 $Events = GenEvents$, analyse the current screen to generate the set of test events
5 $\{event(\alpha) \mid event$ is a UI event, α is an assignment$\}$;
6 **foreach** $event(\alpha) \in Events$ **do**
7 Experiment $event(\alpha)$ on $App \rightarrow$ new screen $Inew$;
8 Analyse $Inew \rightarrow$ assignments act_2, wp_2, wo_2;
9 $ph_2 = \mathcal{P}h\_Deposit(q, act_2, wp_2, wo_2)$;
10 $q_2 = (act_2, wp_2, wo_2, end := null, ph_2)$;
11 **if** *$Inew$ reflects a crash or there exists $[q'] \in Q/\sim$ such that $q_2 \in [q']$* **then**
12 Add a transition $q \xrightarrow{event(\alpha)} q_2 = (act_2, wp_2, wo_2, end := true, ph := 0)$ to $\rightarrow_{\mathcal{P}}$;
13 **else**
14 Add a transition $t = q \xrightarrow{event(\alpha)} q_2 = (act_2, wp_2, wo_2, end := false, ph_2)$ to $\rightarrow_{\mathcal{P}}$;
15 $Q/\sim = Q/\sim \cup \{[q_2]\}$;
16 Add the task $(Explore(q_2, p.t))$ to the task-pool;
17 **if** *Backtrack(q, p)==false* **then**
18 Reset and Execute App by covering the sequence of actions p;
19 **foreach** $q \xrightarrow{event(\alpha)} q_2$ *such that $ph_2 > ph$ and $Explore(q_2, p_2)$ in the task-pool)* **do**
20 Experiment $event(\alpha)$ on App;
21 Take and Execute $Explore(q_2, p_2)$;
22 **if** *Backtrack(q, $q2$)==false* **then**
23 End;

Explore has to reset the application and to incrementally replay the actions of the path p before experimenting the state q.

Once the exploration of the state q is finished, the *Explore* procedure now simulates an ant which pursues its trail. The application exploration is indeed extended at the state q_2, on condition that q_2 is directly reachable and that q_2 contains an assignment of the ph variable higher than the one found in q (line 19). If there is no such state q_2 or if the back mechanism cannot be applied, then the *Explore* procedure terminates. The current thread goes back to the task-pool, and picks out a task previously built by any other thread (in Algorithm 1).

Remark 4 *MCrawlT supports both deterministic and non-deterministic applications. For sake of readability, we have concealed this feature in the previous algorithms in the line "Reset and Execute App by covering the action sequence of p". Given a task $Explore(q, p)$, when the path p is replayed to reach the state q, MCrawlT continuously checks if the arrival state is the one expected in the path p or a new state q'(indeterministic case). If this is a new state, MCrawlT adds a new transition leading to q' and carries on the execution of p.*

Block	States
$B2$	$q6, q7$
$B1$	$q0\_1, q2\_1, q2\_2, q6\_1, q6\_2, q7\_1, q7\_2, q8\_1$

Figure 7: Blocks of states of the minimised PLTS

The above algorithms rely upon some procedures that are summarised below:

1) PLTS minimisation: We have chosen a bisimulation minimisation technique [18] to make minimised PLTSs. Given a PLTS $\mathcal{P}$, this technique offers the strong advantage to generate a minimised model $\mathcal{MP}$, which is behavioural equivalent to $\mathcal{P}$. In short, this algorithm constructs the state sets (blocks) that are bisimilar equivalent (every state can fire the same actions and the arrival states have to be bisimilar again). A detailed algorithm can be found in [18]. The time complexity of this minimisation technique is also reasonable (proportional to $\mathcal{O}(mlog(n))$ with m the transition number and n the state number).

Figures 4(d) and 7 depict the minimised PLTS obtained with the *Ebay Mobile* application. Some locations are now grouped into blocks. All the final states are bisimilar and grouped into the block $B1$. Furthermore, the states $q6$ and $q7$ are grouped into the Block $B2$ because the same action sequences leading to bisimilar states can be executed from both $q6$ and $q7$.

2) Test event generation : The *Explore* procedure calls *GenEvents*, which constructs test events expressing how to interact with screens. Since this part is already presented in [1], we only briefly introduce it here.

Our algorithm generates a set of test events of the form $\{event(\alpha) \mid event$ is an event, α is an assignment$\}$. It starts collecting the events that may be applied on the different Widgets of the current screen. Then, it constructs assignments of the form $\{w_1.value := v1, ..., w_n.value := vn\}$, with $(w_1, ..., w_n)$ the list of editable Widgets found on the screen and $(v_1, ..., v_n)$ a list of test values.

Instead of using random values, we propose to use several data sets, which can be completed before starting the exploration algorithm (the algorithm does not ask for values) The first one, denoted $User$, is completed with values provided by users. If required, this set should hold the logins and passwords needed to access to the application features relative to user accounts. This implies that the user knows some features of the application. To reduce the test event set, if a user value is devoted to some specific Widgets, this value can be accompanied with Widget names.

The set RV is composed of values well known for detecting bugs, e.g., String values like "&", "", or null, completed with random values. A last set, denoted $Fakedata$, is composed of fake user identities. An identity gathers a list of parameters $(p_1, ..., p_m)$, such as (name, age, email, address, gender), which are correlated together to form realistic identities. Both $User$ and RV sets are segmented per type

Procedure Backtrack

1 Procedure $Backtrack(q = (act, wp, wo, end, ph), q_2)$;
2 **if** *the back mechanism is available* **then**
3 Call the back mechanism $\rightarrow$ screen $INew$;
4 Analyse $Inew \rightarrow$ assignments act', wp', wo';
5 **if** $act \neq act'$ *or* $wp \neq wp'$ *or* $wo \neq wo'$ **then**
6 return false;
7 **else**
8 Add a transition $t = q_2 \xrightarrow{back(\alpha)} q$ to $\rightarrow_P$;
9 return true;
10 **else**
11 return false;

(String, Integer, etc.). During the analysis of the current screen, we collect the types and names of the Widget properties. Then, we search for the largest subset of properties that form an identity with respect to the parameters of $Fakedata$ (e.g., name, age, email, address, gender). We obtain a list of Widget properties $(w_1, ..., w_n)$ that we bind with a set of value lists extracted from $Fakedata$. For instance, if two Widgets called *name* and *email* are found, the fake identities of $Fakedata$ are parsed to remove the undesired parameters and to return a set of identities composed only of a name and an email. Each remaining Widget property is associated to the set $User \cup RV$. For instance, if an editable Widget takes String values, we bind this Widget with the set $String(User \cup RV)$. Now, given a list of Widget properties $(w_1, ..., w_n)$, we have a corresponding list of value sets $(V_1, ..., V_n)$. It remains to generate a set of assignments of the form $\alpha = \{w_1.value := v1, ..., w_n.value := vn\}$. Instead of computing the Cartesian product of $(V_1, ..., V_n)$, we adopted a Pairwise technique [20] to build these assignments. Assuming that errors can be revealed by modifying pairs of variables, this technique strongly reduces the coverage of variable domains by constructing discrete combinations for pairs of parameters only.

Finally, every UI event $event$ is associated to an assignment $\alpha = \{w_1.value := v1, ..., w_n.value := vn\}$, which is added to the set $Events$ and returned to the *Explore* procedure.

3) Call of the back mechanism: Based upon preliminary studies, we observed that the back mechanism does not always allow to go back to the previous state of an application. Actually, this mechanism is sometimes considered as an event allowing to reach a new application state. As a consequence, we always check whether the state, reached after calling this mechanism, is the expected one. The pseudo-code is given in the *Backtrack* procedure.

This procedure calls the back mechanism to undo the most recent action if available and to go back to the state q (line 3). A new screen is observed, and *Backtrack* checks whether this screen is equivalent to the expected one, modelled with q (we compare their Widget properties). If we observe a state different from q or if the back mechanism is not available, *Backtrack* returns "false" (line 6). On the contrary, a transition $q_2 \xrightarrow{back(\alpha)} q$ is added to $\mathcal{P}$ and the procedure returns "true" (line 9).

B. Exploration strategies

Different strategies can be applied to cover an application. These are mainly implemented by means of the $Ph\_Deposit$ procedure which is called to return pheromone amounts but also with the task-pool paradigm. Independently of the chosen strategy, the threads, which are executed to explore an application, always pick out the first task of the task-pool composed of a state having the highest pheromone amount. The task-pool is implemented as an ordered list in descending order.

We succinctly present how to implement some of strategy examples below:

- **BFS strategy:** the classical breadth-first search strategy is the easiest one to put in practice. Indeed, our algorithm is tacitly based upon it. Whenever a new state q_2 is built, it is only needed to set its pheromone amount to 0. In our algorithm, a state is tested in a breadth-wise order and each new task $Explore(q_2, p_2)$ composed of a new state q_2 to visit, is added to the task-pool. The threads shall only take the tasks in the task-pool in the same order as they have been submitted. As a consequence, the PLTS $\mathcal{P}$ is conceived in breadth-first order;
- **DFS strategy:** the depth-first search strategy can be implemented as follows: the initial state q_0 is initialised with a pheromone amount equal to 0. Afterwards, whenever a new state q_2 is detected from another one q, it is completed with the pheromone amount found in q increased by 1. In this case, the next task $Explore(q, p)$ chosen by a thread shall be composed by the last detected state. Tacitly, a DFS strategy is followed;
- **Crash-driven strategy:** the number of detected bugs could also be considered in a strategy: when the number of bugs detected from the states of a path p is higher than the one detected from the states of another path p', it may be more interesting to continue to cover the former for trying to detect the highest number of application defects. We call this strategy crash-driven exploration. This can be conducted by initialising the pheromone amount to 0 in q_0. Next, given a task $Explore(q, p)$, whenever a new state q_2 is detected, it is completed with a pheromone amount equal to the number of bugs detected from all the states of the path p;
- **Semantics-driven strategy:** this kind of strategy denotes an exploration guided by the recognition of the meaning of some Widget properties (text field values, etc.). Here, the pheromone deposit mainly depends on the number of recognised Widget properties and

on their relevance. It is manifest that the semantic-driven strategy domain can be tremendously vast. For e-commerce applications, the login step and the term "buy" are usually important. A strategy example could be then conducted as follows: an authentication process is detected when a text field Widget has the type "passwdtype". In this case, the pheromone amount considered is set to 10, otherwise it is equal to 1. When a Widget name is composed of the term "buy", the pheromone amount added in a new state could be equal to 5, etc.

Many other strategies could be defined to meet the user requirements. Some of them could be defined to target specific application states or features. Others could be conceived in accordance with the intended usage of the inferred models. For instance, if models are later used for generating security test cases, the exploration strategy should be defined to cover the most sensitive features of the application. Other criteria could also be considered, e.g., the number of Widgets found in screens. Furthermore, the previous strategies could also be mixed together.

The PLTS of Figure 4(c) is built with a DFS strategy. Our algorithm starts by visiting the state $q0$ which holds a pheromone amount equal to 0. The actions $a0$ to $a5$ lead to new screens and states $q0\_1, q1, ..., q5$, which have a pheromone amount equal to 1 and have to be explored. Here, the state $q1$ is chosen since it is the first not final state encountered during the exploration of $q0$ and has the highest pheromone amount. From $q1$, the execution of actions leads to new states, e.g., $q6$, $q7$, or $q8$. These states have a pheromone amount equal to 2. The next state having the highest pheromone amount is $q6$. Therefore, this one is explored, and so on.

C. Code coverage enhancement, Exploration Algorithm 2

After the evaluation of the previous algorithm, we observed that the code coverages obtained with some applications was lower than expected. After investigation, we discovered that *MCrawlT* was actually unable to launch some specific features of these applications. As a consequence, several screens were not displayed and explored, which explains low code coverages. For instance, a text editor can delete a document if and only if a document is available. At the moment, no testing tool is able to automatically deduce such a scenario since it belongs to the logic of the application.

To solve an aspect of this problem and to enhance code coverage, we propose a straightforward and general solution that tries to bypass the blocking features in order to deeper explore an application. Intuitively, it is technically possible with Mobile applications to directly instantiate any Activity instead of the initial ones. Doing this sometimes allows to bypass a blocking Activity that cannot be automatically tested. The pseudo-code of this solution is given in Algorithm 2. The latter tries to directly launch every Activity of an application instead of only considering the initial one to infer models. We do not provide the details of the *Explore* procedure since it only requires slightly modifications. For an application App, this algorithm tries to instantiate each Activity and builds a PTLS for each. We obtain the PLTS set $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ and the respective set of minimised PLTS $\{\mathcal{MP}_1, ..., \mathcal{MP}_n\}$. The algorithm starts by analysing App and extracts its Activity list; act_1 represents the initial Activity of App. As in the previous algorithm version, a first PLTS $\mathcal{P}_1$ is generated from the initial Activity act_1 (line 1). Then, each Activity act_i is launched, and a corresponding PLTS $\mathcal{P}_i$ is built (line 3). But the algorithm is designed to inspect new states only, i.e., to prevent from exploring several times a state early encountered during the generation of another PLTS. To do so, the state equivalence classes are now kept in the set EQ, which is used all along the execution. Given a model $\mathcal{P}_i (i > 1)$ under construction, the exploration of a state q is done in the *Explore* procedure if and only if q has not been previously encountered in one of the previous PLTSs. In other words, q is marked as final if q belongs to an equivalence class of EQ.

With this algorithm, we switch the initial Activity to start an application from different entry-points and to potentially scan deeper an application. This process does not always deliver the intended outcomes though, since an Activity, which was not designed to be launched at the beginning of the application, may crash. We show in Section VI that this algorithm achieves better code coverage on 1/3 of the experimented applications.

Algorithm 2: Mobile application exploration v2

input : Application App
output: PLTS $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$, $\{\mathcal{MP}_1, ..., \mathcal{MP}_n\}$

```
// initialisation performed by one thread only
1  Analyse App → list of activities LAct = {act_1, ..., act_n}, list of
   PLTSs {P_1, ..., P_n} with act_1 this root Activity of App;
2  EQ = ∅;
3  foreach Activity act_i in LAct such that act_i has not been
   encountered do
4      Start the application App and Launch act_i;
5      Analyse the current screen → Widget property lists wp, wo;
6      Initialise ph_0;
7      Initialise PLTS P_i with
       q0_{P_i} = (act_i, wp, wo, end := false, ph_0);
8      EQ = EQ ∪ {[q0_{P_i}]};
9      Add (Explore(q0_{P_i}, p = ∅), P_i) to the task-pool;
10     while the task-pool is not empty do
11         Take a task (Explore(q, p, P_i)) such that
           q = (act, wp, wt, end, ph) includes the highest
           pheromone amount ph;
12         Reset and Execute App by executing the sequence of
           actions of p;
13         Explore(q, p, P_i);
       // code executed by one thread
14     MP_i := Minimise(P_i);
```

D. Algorithm complexities and termination

Both Algorithms 1 and 2 run in linear time. Theoretically, for an application App, the number of states to visit may be infinite. But our algorithm covers one state per equivalence class. The number of equivalence classes is finite (Proposition 3), hence, the algorithm is finite. If we denote the number of states and transitions by N and M, Algorithm 1 has a complexity proportional to $\mathcal{O}(M+N+MN+Mlog(N))$. Indeed, the *Explore* procedure covers every transition twice (one time to execute an event and one time to go back to the previous state), and every state is processed once. Hence, the complexity should be proportional to $O(M+N)$. But, sometimes the back mechanism is not available. In this situation, the application is reset and the event sequence of a path p is executed from the initial state $q0$. This path is at worst composed of M transitions and, in the worst case, this step is done for every state with a complexity proportional to NM. Furthermore, the minimisation technique has a complexity proportional to $\mathcal{O}(Mlog(N))$ [18].

More precisely, the number of states N is at most equal to $2m^n$ with m is the maximum number of values that any Widget property can take during the testing of App, and n the number of Widget properties in WP. Indeed, the number of equivalence classes is bounded to m^n (Proposition 3). Additionally, a state has either an assignment $end := true$ or $end := false$ (2 further possibilities). Regarding the number of transitions M, it is finite and depends on the number of test events executed on the application. If each state is at most tested with e events and has k editable Widgets that are iteratively fulfilled nb times with test values, then M is equal to $N * e * nb^2$ (nb^2 is the maximum number of test value tuples returned by the Pairwise technique [20]).

Algorithm 2 is designed as Algorithm 1 except that it infers one model for each Activity of an application App. The Activity number, denoted K is always finite, therefore Algorithm 2 terminates as well. It complexity is proportional to $\mathcal{O}K(M+N+MN+Mlog(N))$.

VI. Implementation and Evaluation

A. Implementation for Android Applications

With the collaboration of the Openium company, we have implemented our solution in a prototype tool, called *MCrawlT* (Mobile Crawler Tool). *MCrawlT* is publicly available in a GitHub repository (https://github.com/statops/MCrawlerT), and is accompanied with a detailed user guide. The tool is specialised for Android applications: in short, these applications are typically Mobile GUI applications built over a set of reusable components. For instance, Activities are components that display screens whereas Service components are used to call remote servers.

As detailed in the above sections, *MCrawlT* infers models from Android applications, it reports code coverage percentages and execution times. Besides, it tries to detect bugs with stress testing (use of values known for revealing bugs such as unexpected values (wrong types), execution of large random event flows on screens). It reports the detected bugs, and generates test cases to replay them. Finally, it displays lightweight or complete storyboards (graphs of screen shots) to simplify the understanding of the application behaviours. Figures 8 and 9 depict two storyboard examples derived from the Ebay Mobile application with different execution times.

Figure 8: Ebay Mobile storyboard

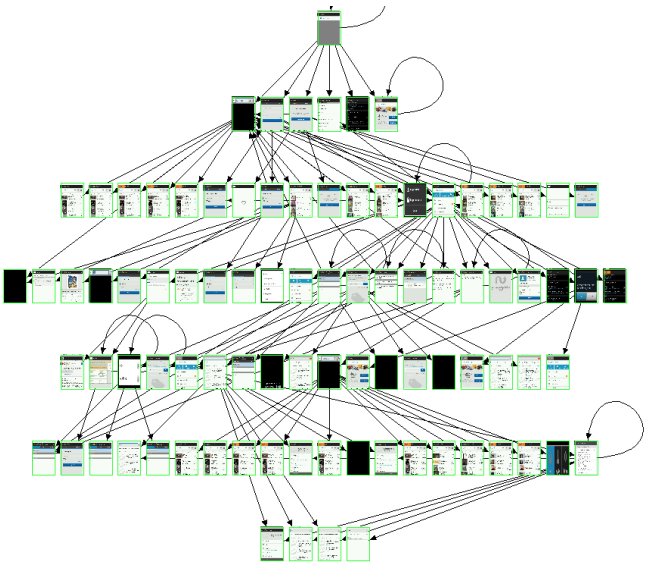


Figure 9: Ebay Mobile storyboard 2

MCrawlT expects packaged Android applications or source projects, a testing strategy and a delay for testing (this delay can be set to some seconds up to several days). A user may add some specific test data (login, password, etc.) and prepare local and/or remote databases.

```
1 public class DFSStrategy extends AntStrategy {

3 public int getRank(State st, ScenarioData path) {
   if (st != null) {
5    State last = path.get(path.size-1));
     return last.getPh()+1;
7    }
   else return defaultRank;
9 } }
```

Figure 10: Implementation of the DFS strategy

MCrawlT is composed by two main modules. *MCrawlT-Desktop* corresponds to Algorithms 1, 2 and essentially aims to construct PLTS and to manage the task-pool. The second module, *MCrawlTMobile*, corresponds to the *Explore* procedure. It is deployed on the smartphone side to exercise application screens and to generate PLTS transitions. The second module is implemented using the testing framework *Robotium* [21]. Robotium is used to extract the Widget properties found in screens. It also provides functionalities for editing Widgets and simulating user events (click, scroll). Additionally, we have extended the instrumentation package of Android (InstrumentationTestRunner class) to detect and observe application crashes, and periodically compute the code coverage percentage by means of the tool *Emma* [22]. The communication between the modules is ensured by the *Android Debug Bridge (adb)* tool, which is available in the Android tool kit. *MCrawlT* can exercise applications in parallel by launching several *MCrawlerTMobile* modules on emulators or smartphones (Android versions from 2.3 to 4.2.2).

MCrawlT supports the strategies presented in Section V-B and can be upgraded with additional ones. A strategy is implemented in a Java class inherited from the class "AntStrategy", which must have a method *public int getRank(State q, ScenarioData path)*. This one returns the pheromone amount which is put down in a state q. Figure 10 illustrates the Java code used to implement the DFS strategy.

B. Limitations

The *MCrawlT* implementation has three main limitations:

- remote servers cannot be reset, so the tool violates an assumption of the algorithm related to the application environment reset. This limitation can be eliminated by mocking remote servers. This can be done with the SOAPUI framework [23];
- *MCrawlT* supports the following UI events: click and scroll. This is a limitation imposed by *Robotium*. But this tool is updated continuously, therefore, more events should be available in the future;
- in the paper, we focus on UI events but Android proposes a set of system events (sms calls, battery notifications, etc.). We do not consider them yet as inputs.

C. Empirical Evaluation

In this section, we evaluate *MCrawlT* on several real-world Android applications. We chose to compare the effectiveness of several recent tools in terms of execution time, code coverage and crash detection. We tried executing the following tools *Monkey* [11], *Guitar* [19], *AndroidRipper* [7], *SwiftHand* [15] and *Dynodroid* [9]. The others are not available. Unfortunately, we faced many difficulties to use some of them. In summary, we do not know how *Guitar* works with Mobile applications due to lack of documentation; we were unable to launch *AndroidRipper*; *SwiftHand* works well with the proposed examples but, for new applications, source codes need to be instrumented (with a lot of classes) and we do not know how to do this.

In this context, and to avoid any bias, we chose to apply our tool, *Monkey* and *Dynodroid* on all the applications whose source code is available and taken for experimentation in the papers [7], [15], [9]. We have also taken the applications and experimental results found in [8] although the corresponding tool *Orbit* is not available. This corresponds to 30 applications. It is important to note that *Monkey* is taken as a reference in most of the papers dealing with Android testing. Thereby, our results can be compared with other studies related to Android testing.

1) Code coverage and execution time: We compare here the effectiveness, relating to code coverage and execution time, of *MCrawlT* with the other recent Android testing tools. Most of them explore these applications in an in-depth manner. So, *MCrawlT* was executed only with this strategy to carry out a fair comparison.

Figure 11 reports the percentages of code coverage obtained with the different tools on 30 applications with a time budget of three hours. If we do a side by side comparison of *MCrawlT* with the other tools, we observe that *MCrawlT* provides better code coverage than *Monkey* for 23 applications, than *SwiftHand* for 29 applications, than *Orbit* for 29 applications, and than *Dynodroid* for 24 applications. In comparison to all the tools, *MCrawlT* provides better code coverage with 20 applications, the coverage difference being higher than 5% with 14 applications and higher than 10% with 10. This comparison is more explicitly given in the radar chart of Figure 12, which depicts the code coverage percentages obtained with *Monkey*, *MCrawlT* and *Dynodroid*. When *Monkey* is confronted with all the other tools, it offers better results for 5 applications and *Dynodroid* for 3 applications.

Consequently, these results show that *MCrawlT* gives better code coverage than each tool taken one by one and overall offers good results against all the tools on half the applications. Figure 12 clearly illustrates this claim.

Figure 11 shows that the code coverage percentage obtained with *MCrawlT* is between 25% and 96%. We manually analysed the 10 applications that provide the lower code

Application	Mon key	Orbit	Gui tar	And. Rip- per	MC rawlT	Swift Hand	Dyno droid
NotePad	60	82			88		
TippyTipperV1	41	78			79		48
ToDoManager	71	75	71		81		34
OpenManager	29	63			65		
HelloAUT	71	86	51		96		76
TomDroid	46	70		40	76		42
ContactManager	53	91	71		68		28
Aardict	52	65		27	67		51
Musicnote	69				81	72.2	47
Explorer	58				74	74	
Myexpense	25				61	41.8	40
Anynemo	61				54	52.9	
Whohas	58				95	59.3	65
Mininote	42				26	34	39
Weight	51				34	62	56
TippyTipperV2	49				74	68	12
Sanity	8				26	19.6	1
Nectdroid	70.7				54		68.6
Alogcat	66.6				66		67.2
ACal	14				46		23
Anycut	67				71		69.7
Mirrored	63				76		60
Jamendo	64				46		3.9
Netcounter	47				56		70
Multisms	65				73		77
Alarm	77				72		55
Bomber	79				75		70
Adsdroid	72				83		80
Aagtl	18				25		17
PasswordFor Android	58				61		58

Figure 11: Code coverage comparison (in %)

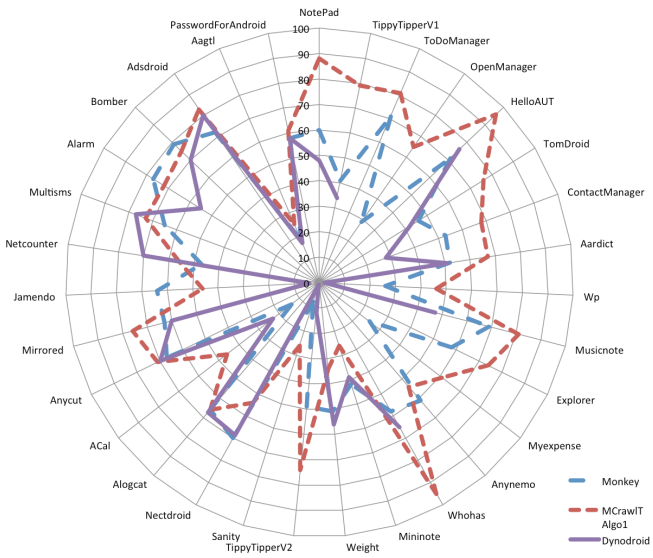


Figure 12: Code coverage comparison (in %)

coverage percentages with *MCrawlT* to identify the causes behind low coverage.

These can be explained as follows:

- specific functionalities and unreachable code: several applications are incompletely covered either on account of unused code parts (libraries, packages, etc.) that are not called by the application, or on account of functionalities difficult to start automatically. For instance, at least one stored audio file is required for *OpenManager* before testing the functionalities related to the audio file management,
- unsupported events: Several applications, e.g., Nectdroid, Multism, Acal or Alogcat chosen for experimentation with *Dynodroid* take UI events as inputs but also system events such as broadcast messages from other applications or from the Android system. Our tool does not support these events yet. Moreover, MCrawlT only supports the event list also supported by the testing tool *Robotium* (click, scroll). The long click event does not belong to this list but it is used in some applications (Mininote and Contactmanager). In contrast, *Orbit* supports this event and therefore offers a better code coverage with the application Contactmanager.

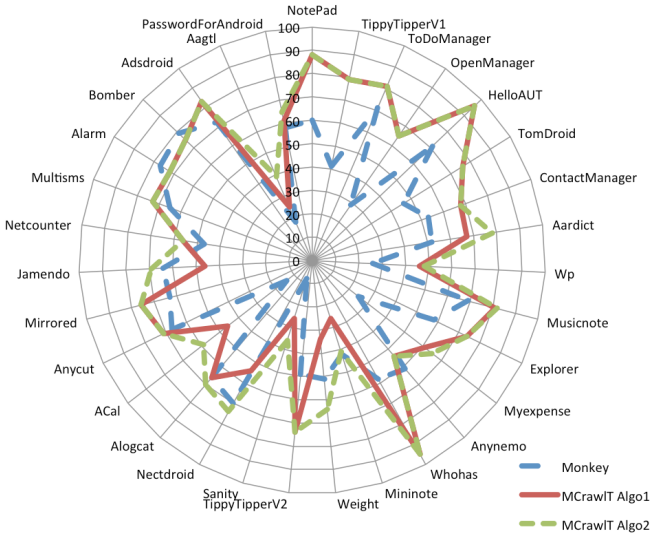


Figure 13: Code coverage comparison (in %)

We also experimented the 30 Android applications with the second version of our algorithm, which tries to infer models for every Activity (Algorithm 2). We kept the same time budget of three hours. The radar chart of Figure 13 gives the code coverage percentages obtained with *Monkey* (for comparison purposes), *MCrawlT* Algorithm 1 and Algorithm 2. It illustrates that our extended algorithm visibly offers better code coverage. More precisely, 13 applications are more covered and 8 have a code coverage increased by 10 %. With some applications the code coverage difference become significant. For instance, we observe a code coverage increased by 23 % with Jamento and by 30 % with Weight. In comparison to all the tools, *MCrawlT* provides now better code coverage with 25 applications.

Regarding execution time, the evaluated tools work differently. *Monkey* and *Dynodroid* take a number of events, and perform fuzzy testing independently of the application

Application	Orbit	Guitar	Android Ripper	MCrawlT	Swift Hand
NotePad	102			268	
TippyTipperV1	198			251	
ToDoManager	121	194		551	
OpenManager	480			696	
HelloAUT	156	117		106	
TomDroid	340		529	235	
Contact Manager	125	194		233	
Aardict	124		694	580	
Musicnote				10696	10800
Explorer				10800	10800
Myexpense				10800	10800
Anynemo				10800	10800
Whohas				9260	10800
Mininote				8230	10800
TippyTipperV2				1556	10800
Weight				10800	10800
Sanity				10800	10800
Nectdroid				8120	
AlogCat				10800	
ACal				10800	
Anycut				8037	
Mirrored				6020	
Jamendo				10800	
Netcounter				10800	
Multisms				10800	
Alarm				10040	
Bomber				4800	
Adsdroid				10800	
Aagtl				920	
Password ForAndroid				10800	

Figure 14: Execution time (in seconds)

Application	MCrawlT	Monkey	Dynodroid	Android Ripper
WordPress	63	3		37
Notepad	5			
TomDroid	7	1		14
Mirror	25	3		
Mininote	2			
Aagtl	1		2	
PasswordForAndroid	1		1	
Sanity	5	1	1	
Aardict	2			

Figure 15: Application crash detection

coverage. We set an event number sufficiently high so as to let the tools perform testing during three hours or more (more than 10,000 events for *Monkey* and more than 600 events for *Dynodroid*). The other tools explore applications with a delay of three hours or until all the application states are explored. Figure 14 reports the execution times obtained with the second category of tools (in seconds) on the same application list. These results reflect the fact that *MCrawlT* is comparable to the others despite using strategies, state equivalence classes and a state minimisation technique. For small applications (first eight lines), we obtain roughly similar time executions as *Orbit*, *Guitar* and *AndroidRipper*. *SwiftHand* always need more than three hours to explore applications, whereas 18 applications are completely explored with *MCrawlT* in less time.

2) Crash detection: MCrawlT, *Monkey*, *Dynodroid* and *AndroidRipper* also detect application crashes. Figure 15 reports the 9 applications that crashed while testing with one of third first tools. We also added the empirical results obtained with *AndroidRipper* found in [7]. Figure 15 exposes genuine bugs only. We manually ascertained test reports to eventually remove false positives such as emulator misbehaving. We kept only the Exceptions that cause the termination of the applications such as *NullPointerException*. On the 30 applications, *MCrawlT* revealed that 9 of them have bugs and detected all the applications also found with *Monkey* and *Dynodroid*. We deduce from these results that our tool outperforms the others in automatic crash detection. *MCrawlT* does stress testing like *Monkey* and *Dynodroid* but it also uses values known for detecting bugs. This probably explains the better performance.

3) Impact of the strategy choice and parallelism gain: To illustrate the benefits of using different strategies, we applied on the Ebay Mobile application, the DFS strategy exposed in the previous section and a semantics-driven strategy. This strategy aims to target the account management part of the application and was applied by deposing a higher pheromone amount in states including Widgets of type ”passwdtype” or Widget properties composed of the terms ”account” or ”sign in”.

For readability and comparison purposes, we illustrate in Figure 16 a simplified graph showing the visited Activities with the DFS strategy. The application is explored independently of the meaning of the Widgets and the Activities. Here, *MCrawlT* has mostly covered the ”Refine search” feature of the Ebay Mobile application. Figure 17 illustrates the simplified graph achieved after applying the second one. Here, the Activity $SignIn$, allowing to log in to user accounts, was firstly visited instead of the Activity *RefineSearch*. Then, the second strategy has guided the exploration on the Activity *SavedSellerList*, which allows to manage the favourite seller list and on *SellItem*, which shows the sold items. With the same time budget, the account management part of the application has been explored with the second strategy instead of the ”Refine search” feature. As a consequence, since security vulnerabilities on the user account management affect users and may lead to serious consequences, this strategy makes the generated PLTS more interesting to later analyse the security of the application.

The strategy choice also impacts the time required to explore an application. Figure 18 shows the execution times (in seconds) obtained with two strategies for completely exploring 10 applications. *MCrawlT* were applied with a DFS strategy (1 thread using 1 Android emulator), a BFS (with 1 and 3 threads in parallel). These results show that 7 out of 10 applications are more rapidly covered with BFS traversing. For instance, with *toDoManager*, using the BFS strategy instead of the DFS one, reduces the exploration time by 140 seconds because all of its Activities are directly accessible from the initial one. Actually, when a user has knowledge of the code structure or of how the Activities

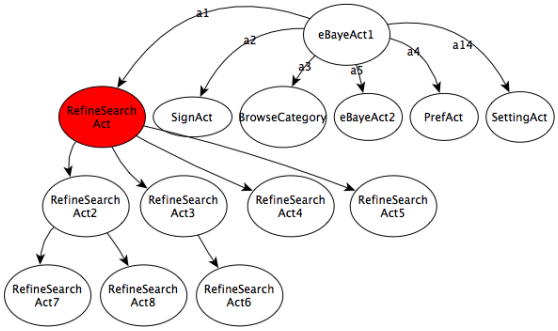


Figure 16: Ebay Mobile simplified graph obtained with a DFS strategy

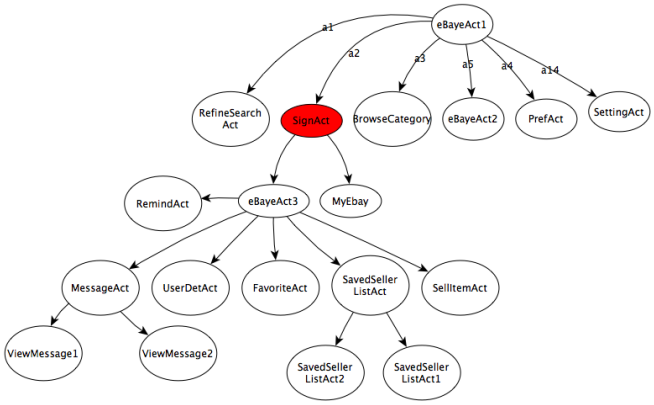


Figure 17: Ebay Mobile simplified graph obtained with a semantics-driven strategy

Application	DFS(1)	BFS(1)	BFS(3)
NotePad	268	310	175
TippyTipperv1	251	210	110
ToDoManager	551	410	210
OpenManager	696	560	489
HelloAUT	106	216	201
TomDroid	235	256	196
ContactManager	233	216	135
Bomber	6120	4800	3100
Mirrored	6690	6020	4090
Nectdroid	10650	8120	5020

Figure 18: Execution time with different strategies (in seconds)

are composed together, he can choose the most appropriate strategy to speed up the exploration.

Figure 18 also shows that the parallelization of our algorithm is effective. With three emulators, the execution time is always reduced. For instance, the parallel exploration of TippyTipperV1 is achieved with a time almost divided by two.

All these experimental results on real applications tend to show that our tool is effective and leads to substantial improvements in the automatic testing of Mobile applications. Indeed, on the 30 applications taken for evaluation in [7], [15], [9], *MCrawlT* gives better code coverage for 25 applications with the same time budget and detects more bugs. Furthermore, different exploration strategies can be applied to directly target the most relevant application features.

VII. Conclusion

Automatic testing of GUI applications is an interesting solution that complements other testing techniques, e.g., Model-based testing. This approach may be used to generate partial models, which can be later completed or reused for test case generation. This paper brings some original contributions by proposing: 1) a formal model definition that helps limit the application exploration by segmenting state space domains into finite sets of equivalence classes, 2) the use of exploration strategies to cover applications by applying the ACO technique, 3) a code coverage enhancement method which infers sets of models.

The evaluation of *MCrawlT* against other recent tools shows that our approach can be used in practice. It automates testing tasks that users usually consider tedious. Furthermore, it generates models and storyboards that can be used for model analysis and comprehension aid. *MCrawlT* also provides good code coverage quickly and detects more bugs than those exposed by the other tools.

The initial purpose of this work was to generate partial models given to an automatic security testing method for Mobile applications. Based upon this framework, we intend to design this security testing method by developing these future research directions: 1) define an exploration strategy in order to automatically detect the highest number of security issues while the model generation, 2) devise or reuse verification methods on inferred models to detect security vulnerabilities, 3) generate test cases to extend the inferred models from some specific states in an attempt to expose further security vulnerabilities.

References

[1] S. Salva and S. R. Zafimiharisoa, “Model reverse-engineering of mobile applications with exploration strategies,” in *The Ninth International Conference on Software Engineering Advances, ICSEA 2014*, Nice, France, 10 2014, pp. 396–403.

[2] A. Memon, I. Banerjee, and A. Nagarajan, “Gui ripping: Reverse engineering of graphical user interfaces for testing,” in *Proceedings of the 10th Working Conference on Reverse Engineering*, ser. WCRE ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 260–269. [Online]. Available: http://dl.acm.org/citation.cfm?id=950792.951350

[3] A. Mesbah, A. van Deursen, and S. Lenselink, “Crawling Ajax-based web applications through dynamic analysis of user interface state changes,” *ACM Transactions on the Web (TWEB)*, vol. 6, no. 1, pp. 1–30, 2012.

[4] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," *Software Engineering, IEEE Transactions on*, vol. 36, no. 4, pp. 474–494, 2010.

[5] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: a tool for testing web 2.0 applications," in *Proceedings of the Workshop on JavaScript Tools*, ser. JSTools '12. New York, NY, USA: ACM, 2012, pp. 11–15. [Online]. Available: http://doi.acm.org/10.1145/2307720.2307722

[6] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393666

[7] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 258–261. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351717

[8] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated gui-model generation of mobile applications," in *Proceedings of the 16th international conference on Fundamental Approaches to Software Engineering*, ser. FASE'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 250–265. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37057-1_19

[9] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 224–234. [Online]. Available: http://doi.acm.org/10.1145/2491411.2491450

[10] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. Ta, and A. Memon, "Mobiguitar – a tool for automated model-based testing of mobile apps," *IEEE Software*, vol. 99, no. PrePrints, pp. 1–6, 2014.

[11] Google. Ui/application exerciser monkey. Accessed: 2015-03-01. [Online]. Available: http://developer.android.com/tools/help/monkey.html

[12] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '13. New York, NY, USA: ACM, 2013, pp. 641–660. [Online]. Available: http://doi.acm.org/10.1145/2509136.2509549

[13] D. Amalfitano, A. Fasolino, and P. Tramontana, "Reverse engineering finite state machines from rich internet applications," in *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, Oct 2008, pp. 69–73.

[14] M. E. Joorabchi and A. Mesbah, "Reverse engineering ios mobile applications," in *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, ser. WCRE '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 177–186. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2012.27

[15] W. Choi, G. Necula, and K. Sen, "Guided gui testing of android apps with minimal restart and approximate learning," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '13. New York, NY, USA: ACM, 2013, pp. 623–640. [Online]. Available: http://doi.acm.org/10.1145/2509136.2509552

[16] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987. [Online]. Available: http://dx.doi.org/10.1016/0890-5401(87)90052-6

[17] D. Amalfitano, A. Fasolino, and P. Tramontana, "A gui crawling-based technique for android mobile application testing," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, 2011, pp. 252–261.

[18] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Science of Computer Programming*, vol. 13, pp. 13–219, 1989.

[19] B. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "Guitar: an innovative tool for automated testing of gui-driven software," *Automated Software Engineering*, vol. 21, no. 1, pp. 65–105, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10515-013-0128-9

[20] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *Proc. of the 25th International Conference on Software Engineering*, 2003, pp. 38–48.

[21] Robotium, user scenario testing for android. Accessed: 2015-03-01. [Online]. Available: http:/code.google.com/p/robotium/

[22] Emma, a free java code coverage tool. Accessed: 2015-03-01. [Online]. Available: http://emma.sourceforge.net

[23] Soapui. Accessed: 2015-03-01. [Online]. Available: http://www.soapui.org

Developing Heterogeneous Software Product Lines with FAMILE – a Model-Driven Approach

Thomas Buchmann and Felix Schwägerl
University of Bayreuth
Chair of Applied Computer Science I
Bayreuth, Germany
{thomas.buchmann, felix.schwaegerl}@uni-bayreuth.de

***Abstract*—Model-Driven Software Development and Software Product Line Engineering are independent disciplines, which both promise less development effort and increased software quality. While Model-Driven Software Development relies on raising the level of abstraction and automatic code generation, Software Product Line Engineering is dedicated to planned reuse of software components based upon a common platform, from which single products may be derived. The common platform consists of different types of artefacts like requirements, specifications, architecture definitions, source code, and so forth. Only recently, research projects have been started dealing with model-driven development of software product lines. So far, the resulting tools can only handle one type of artefact at the same time. In this paper, requirements, concepts and limitations of tool support for heterogeneous Software Product Line Engineering are discussed. As a proof of concept, an extension to the model-driven tool chain FAMILE is presented, which supports mapping of features to different types of artefacts in heterogeneous software projects at the same time. The added value of the approach is presented by an example product line, which has been developed in a strictly model-driven way using FAMILE.**

Keywords–software product lines; model-driven development; negative variability; feature models; heterogeneity.

I. INTRODUCTION

This article is an extended version of an ICSEA 2014 conference paper [1]. It contains a deeper insight into the tool FAMILE and the adaptations, which enable the management of heterogeneous software product lines. It also presents the relevant ideas using a concrete example.

Model-Driven Software Engineering (MDSE) [2] puts strong emphasis on the development of high-level models rather than on the source code. Models are not considered as documentation or as informal guidelines how to program the actual system. In contrast, models have a well-defined syntax and semantics. Moreover, MDSE aims at the development of *executable* models. The Eclipse Modeling Framework (EMF) [3] has been established as an extensible platform for the development of MDSE applications. It is based on the Ecore metamodel, which is compatible with the OMG Meta Object Facility (MOF) specification [4]. Ideally, software engineers operate only on the level of models such that there is no need to inspect or edit the actual source code, which is generated from the models automatically. However, practical experiences have shown that language-specific adaptations to the generated source code are frequently necessary. In EMF, for instance, only structure is modeled by means of class diagrams, whereas behavior is described by a posteriori modifications to the generated source code.

Software Product Line Engineering (SPLE) [5][6] deals with the systematic development of products belonging to a common system family. Rather than developing each instance of a product line from scratch, reusable software artefacts are created such that each product may be composed from a collection of reusable artefacts — the platform. Commonalities and differences among different products may be captured in a *feature model* [7], whereas *feature configurations* describe the characteristics of particular products by selecting or deselecting features. Typical SPLE processes distinguish between *domain engineering*, which deals with the establishment of the platform as well as the feature model, and *application engineering*, which is concerned with the derivation of particular products out of the product line by exploiting and binding the variability provided by the platform.

To realize variability in SPLE, two distinct approaches exist: In approaches based upon *positive variability*, product-specific artefacts are built around a common core [8][9]. *Composition* techniques are used to derive products. In approaches based on *negative variability*, a *superimposition* of all variants is created — a *multi-variant product*. The derivation of products is achieved by removing all fragments of artefacts implementing features that are *not* contained in the specific feature configuration [10][11]. While approaches based on positive variability typically require new languages, negative variability can be applied to existing ones by means of using preprocessor like tools. Thus, approaches based on negative variability can easily be applied to already existing software artefacts. The tool chain "Features and Mappings in Lucid Evolution" (FAMILE) [12][13], which is used in this paper, belongs to the latter category.

In the past, several approaches have been taken in combining SPLE and MDSE to get the best out of both worlds. Both software engineering techniques consider models as primary artefacts: Feature models [7] are used in SPLE to capture the commonalities and differences of a product line, whereas Unified Modeling Language (UML) models [14] or domain-specific models are used in MDSE to describe the software system at a higher level of abstraction. The resulting integrating discipline, Model-Driven Software Product Line Engineering (MDPLE), operates on a higher level of abstraction compared to traditional software product line approaches operating on the source code level. By this integration, an additional increase in productivity is achieved. In the special case of negative

variability, the platform is provided as a *multi-variant domain model*. The upcoming MDPLE approach has been successfully applied in several case studies, including MOD2-SCM [15], a model-driven product line for software configuration systems.

In this paper, requirements, concepts and limitations of tool support for *heterogeneous* software product lines (HSPLs) are discussed. Here, the term 'heterogeneity' means that (a) artefacts are distributed over multiple resources, (b) the underlying data format of artefacts may differ (e.g., text files or XMI files), (c) in the case of models, the metamodel may vary, (d) artefacts are connected by both explicit and conceptual links, and (e) variability among different resources may be expressed by a shared variability model that uses a common variability mechanism. Based upon these assumptions, several conceptual extensions to MDPLE frameworks are developed, which are implemented in the form of extensions to the tool chain FAMILE as a proof of concept. The practical value of the new approach is shown by developing a heterogeneous product line for editors for *graphs*, which are distinguished by properties such as the editor type (tree or graphical) or the types of graphs that may be edited (weighted, directed, and/or colored graphs, etc.).

The paper is structured as follows: After clarifying the contribution (Section II), the state of the art of homogeneous SPLE tools is outlined in Section III. A brief introduction of the running example is given in Section IV, while Section V explains the new concepts introduced for the support of heterogeneous product lines. In Section VI, the example is revisited in order to demonstrate the heterogeneous extension to the MDPLE tool chain FAMILE on the graph product line, which has been modeled using Eclipse Modeling Technology (EMF and the Graphical Modeling Framework (GMF) [16]). Related work is discussed in Section VII, while Section VIII concludes the paper and outlines future work.

Both the tool chain and the running example project may be retrieved via an Eclipse update site (http://btn1x4.inf.uni-bayreuth.de/famile2/update).

II. State of the Art, Challenges, and Contribution

Heterogeneous software projects consist of a variety of interconnected resources of different types. Different representations may be used for requirements engineering, analysis and design. The generated source code is typically expressed in a general purpose language, e.g., Java, and extended with language-specific – mostly behavioral – components. Furthermore, a software project contains a set of configuration files such as build scripts, which are typically represented in plain text or XML format. In order to adequately handle variability of the overall software project, all these different artefacts need to be subject to variability management.

In its current state, *tool support* for model-driven product line engineering does not adequately address heterogeneous software projects (see Section VII). In particular, the following new challenges arise for SPLE tools:

(a) They should ensure the consistency of *cross-resource links* between different artefacts.

(b) The *level of abstraction* needs to be variable, i.e., the tool should be able to operate both at the modeling and at the source code level.

(c) Different artefacts are based on different formalisms, e.g., metamodels or language grammars. In the special case of models, supporting a mixture of different metamodels requires adequate tool support.

(d) In the case of model-driven engineering, there exist conceptual links between different artefact types. For example, when adding variability to a certain modeling construct, the corresponding generated source code fragment needs to be provided with the same variability information in order to keep the product line consistent.

(e) All artefacts must be handled by a *uniform variability mechanism* (e.g., a common feature model) in order to allow for product configuration in a single step.

In this paper, an approach to heterogeneous SPL development is presented, which advances the state of the art by the following conceptual contributions:

(a) **Multi-resource artefacts** Heterogeneous projects consist of inter-related artefacts created for different development tasks such as requirements engineering or testing. The referential integrity among these inter-related models is maintained during product derivation.

(b) **Heterogeneous artefact types** The approach presented here can handle product lines composed from different kinds of artefacts. Technically, an abstraction from different resource types is conducted by representing them as EMF models.

(c) **Variable metamodels** In the special case of models, the approach presented here does not assume a specific metamodel but allows an arbitrary mixture of models, which may be instances of any Ecore-based metamodel(s).

(d) **Maintenance of conceptual links** The presented approach recognizes dependencies between heterogeneous artefacts even in case they are not modeled explicitly. This is true, e.g., for the concrete Java syntax, which may be provided with variability information, too. Internally, the corresponding concrete syntax fragment is mapped to an abstract syntax tree, which is invisible to the user.

(e) **Common variability mechanism** In the original version of FAMILE, the variability mechanism of feature models has been applied to single-resource EMF models. The presented approach allows for an extension of the product space to almost arbitrary resources. All artefacts are managed by a unique feature model.

These conceptual contributions will be demonstrated by the example of a proof-of-concept implementation that provides an extension to the FAMILE toolchain [12][13]. The extended version of FAMILE can deal with plain text files, XML files, Java source code files, arbitrary EMF models, and further types of resources. This way, variability within complete Eclipse

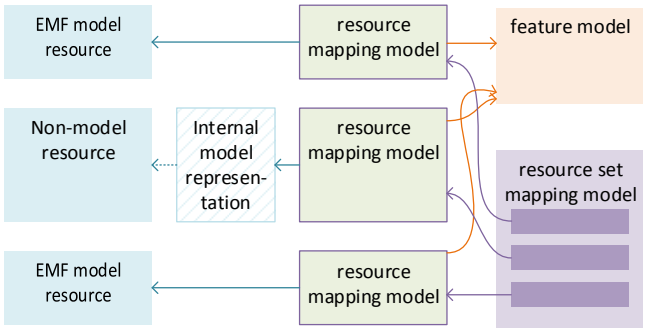


Figure 1. Conceptual mapping of models and non-model artefacts in the presented approach.

projects may be managed. Internally, all artefacts, even plain text and XML files, are represented as EMF models.

For each resource that is subject to variability, a single-resource mapping model (i.e., a model that maintains traceability links between the variability model and the multi-variant domain model) is created, which may be managed by the existing FAMILE core. The consistency between those different mapping models is maintained by an additional *resource set mapping model*. Figure 1 shows how single resource and resource set mapping models are used in order to manage a heterogeneous Eclipse project. In Section VI-D, the interplay between the different resource types is discussed in detail.

III. State of the Art: Homogeneous MDPLE tools

This section provides a brief overview on the state of the art of current tools for model-driven product line engineering. The description is confined to approaches based on negative variability. As one representative, the original version of the FAMILE tool chain [12][13] is presented. FAMILE is tailored towards software product line development processes that distinguish between domain and application engineering [5][6]. *Domain engineering* is dedicated to analyzing the domain and capturing the results in a *feature model*, which describes commonalities and differences thereof. Furthermore, an implementation – the *multi-variant domain model* – is provided as a result of this phase, which is then used during *application engineering* to derive application specific products. Figure 2 depicts the software product line process.

Current MDPLE tools – in particular, FAMILE – support this process by assisting in the following tasks:

1) **Definition of a feature model** At the beginning of the domain engineering phase of the product line life-cycle, the problem domain is analyzed and the commonalities and differences are captured in a *feature model* [7]. For feature models, several extensions such as cardinality-based feature modeling [17] have been proposed.
2) **Creation of the domain model** For the construction of a multi-variant domain model, modelers may use their preferred modeling languages and tools. Most MDPLE approaches only support single-resource models. FAMILE requires that the resulting model is an instance of an Ecore metamodel.

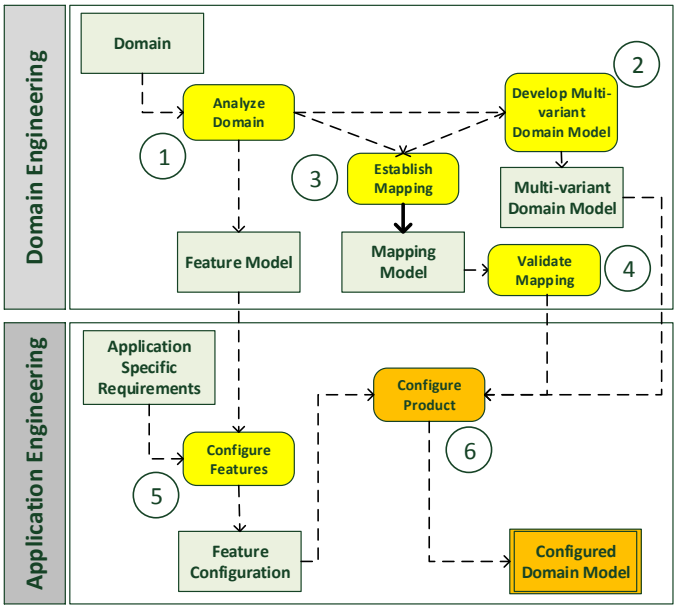


Figure 2. The software product line process supported by the state-of-the-art tool FAMILE.

3) **Mapping features to model elements** In order to define which parts of the domain model realize which feature (or which combination thereof), MDPLE tools provide different mechanisms to map features to model elements. For this purpose, FAMILE includes the Feature to Domain Mapping Model (F2DMM) editor, which supports the process of assigning *feature expressions* – arbitrary propositional formula on the set of features – to particular model elements of a single resource. A feature expression is a logical combination of features, for which FAMILE provides a dedicated textual language (FEL, Feature Expression Language). Modelers can either assign feature expressions by drag-and-drop or by selecting a model element in the editor and textually entering the expression [12].
4) **Ensuring the consistency of the product line** The increasing complexity coming with both the size of the multi-variant domain model and the number of features requires sophisticated mechanisms to detect and repair inconsistencies among the artefacts of the product line. In particular, the consistency between (a) the mapping model and the domain model, (b) the feature model and its corresponding feature configurations, and (c) feature expressions and the feature model, must be ensured. Different approaches are described in [17][18]. FAMILE introduces the concepts of *surrogates* and *propagation strategies* [13] for this purpose.
5) **Definition of feature configurations** As soon as the mapping is complete, MDPLE tools support the creation of *feature configurations*, each describing the characteristics of a member of the software product line. For each feature defined in the feature model, a *selection state* must be provided that determines whether a feature is present in the corresponding

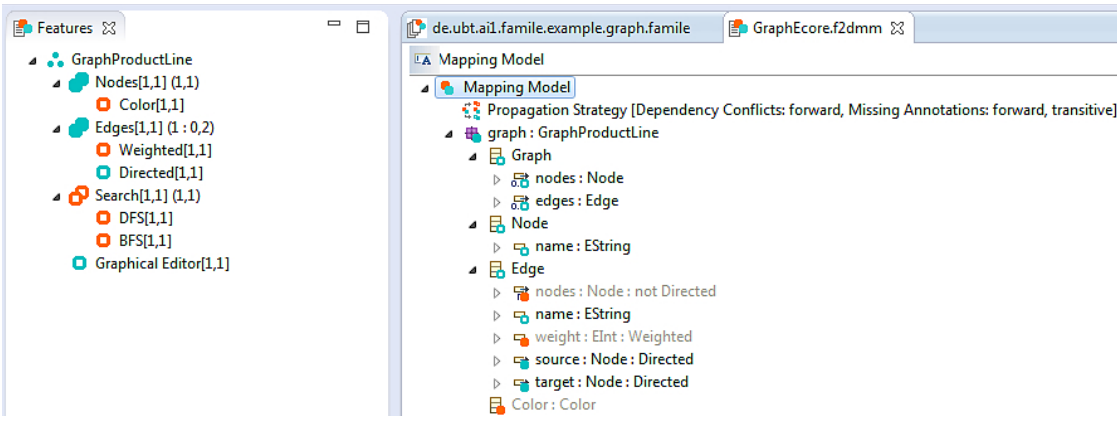


Figure 3. Screenshot of the F2DMM mapping model editor showing the multi-variant domain model of the (homogeneous) graph product line.

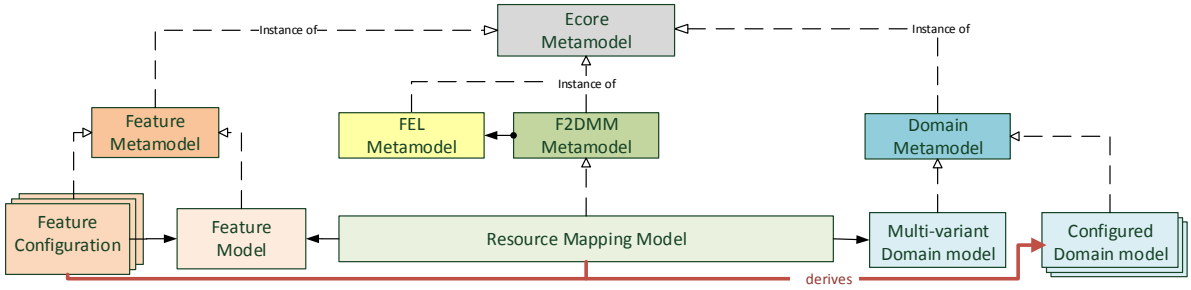


Figure 4. Metamodels and models involved in the original version of FAMILE. Different models are used to map a single-resource multi-variant domain model. All metamodels are based on Ecore.

product.

6) **Product derivation** A specific product can be derived by applying its corresponding feature configuration to the product line. During the derivation process, the multi-variant domain model is filtered by elements whose assigned feature expressions evaluate to false, i.e., the corresponding features are deselected in the current feature configuration. In homogeneous MDPLE tools, the result of this operation is a product-specific single-resource model represented as an instance of the (previously fixed) domain metamodel.

IV. Example: Homogeneous FAMILE Product Line for Graph Metamodels

The following statements refer to the original version of the tool FAMILE as one representative of homogeneous MDPLE tools. Section V demonstrates how heterogeneous project support is added to the tool chain. As a demonstrating example within this paper, the *graph product line* example has been adopted, which is frequently used in research papers because it is easy to understand and its size is rather small [19].

FAMILE itself has been developed using EMF as its technological foundation. A model-driven software product line developed with FAMILE is spread over multiple EMF resources, which are instances of multiple metamodels (cf. Figure 4): Feature models and configurations share a common metamodel that supports cardinality-based feature modeling. The (single-resource) F2DMM mapping model describes how domain model elements are mapped to features. The domain model is an instance of an arbitrary domain metamodel, which is fixed for the mapped resource. It is assumed to be a single-resource entity. The Feature Expression Language (FEL) metamodel describes a textual language for feature expressions [12].

With the F2DMM editor (see Figure 3), the user is assisted in assigning feature expressions to domain model elements. The underlying F2DMM mapping model is constructed automatically and reflects the spanning containment tree structure of the domain model (in this case, the domain model is an Ecore class diagram). Using the reflective EMF editing mechanism [3], the F2DMM user interface emulates the reflective EMF tree editor. Optionally, the user may load an example feature configuration already during the mapping process in order to comprehend how feature expressions are evaluated. The screenshot shown in Figure 3 depicts an example feature configuration in the left pane. Selected features or groups are displayed in cyan, deselected features or groups in orange. The right pane contains the mapping of specific features to artefacts of the multi-variant domain model. Elements are annotated with feature expressions after a colon. The calculated selection states *selected* and *deselected* are represented in cyan and orange.

The example feature configuration shown in Figure 3 represents a directed graph (with uncolored nodes and unweighted edges) that realizes neither depth-first search nor breadth-first search.

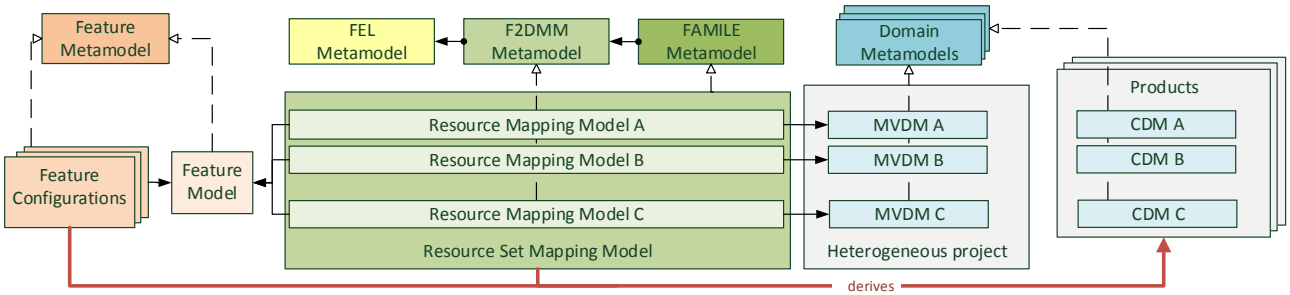


Figure 5. Metamodels and models involved in the extension of FAMILE. Abbreviations: MVDM = multi-variant domain model; CDM = configured domain model.

Considering a (model-driven) software product line as a homogeneous artefact causes a considerable amount of limitations. When referring to the graph example, it is obvious that, although being the core artefact of a model-driven project, the metamodel is not "everything". Using only the metamodel, one is not able to express several behavioral aspects (e.g., the implementation of generated method bodies) or details of representation (e.g., tree or diagram editors), only to name a few. Thus, it is necessary to include further resources into the product line. In the subsequent section, the conceptual and technical prerequisites for heterogeneous SPL support are discussed, before the running example is revisited in Section VI, where the platform will be specified in a greater level of detail. Furthermore, the underlying feature model will be defined more precisely.

V. Support for Heterogeneous Approaches

This section explains how support for heterogeneous model-driven software product lines has been added to the MDPLE tool FAMILE. From a technical point of view, this requires multiple metamodels for the platform and multiple models that describe different artefacts of the product in different stages of the development process (e.g., requirements, design, implementation). As stated in the introduction, it is assumed that all project artefacts may be expressed using EMF.

Figure 5 shows the conceptual overview of the new, heterogeneous version of the FAMILE tool chain. A resource set mapping model is an instance of the FAMILE metamodel and wraps different single-resource F2DMM model instances, which are used for mapping features to the different (heterogeneous) multi-variant domain model instances. A resource set mapping model references a given feature model and one out of an arbitrary number of corresponding feature configurations. Features are mapped to the corresponding domain artefacts by using a separate mapping model per resource.

In Subsection V-A, the new FAMILE metamodel will be presented as the core of the heterogeneous extension. Subsection V-B explains how non-model artefacts are mapped to EMF models, which is a technical necessity in order to manage them in a FAMILE product line. Subsection V-C will present additional user interface components that ease heterogeneous SPL development.

A. The FAMILE Metamodel

The specific requirements of heterogeneous modeling projects have been addressed by the FAMILE metamodel and its corresponding instances, which constitute an extension to the F2DMM metamodel, where models have been considered as self-contained single-resource entities [12]. While this approach works well for projects with only one domain metamodel, it is obvious that heterogeneous projects, e.g., a GMF project, cannot be handled this way. Furthermore, even in non-heterogeneous projects, a model might be split up into different resources to better cope with size and/or complexity. In order to support multiple (EMF-based) resources of potentially different types, the new FAMILE model shown in Figure 6 wraps several instances of the F2DMM metamodel, which still constitutes the core of the extended tool chain.

The FAMILE metamodel (cf. Figure 6) defines a logical grouping of inter-related mapping models. The root element – an instance of ProductLine – defines a number of *global project parameters*, being the references to the used feature model and optionally a feature configuration, as well as a *propagation strategy* (used for automatic detection and resolution of inconsistencies; see [13]). FAMILE takes care that global project parameters are kept consistent within different resource mappings of the same heterogeneous product line.

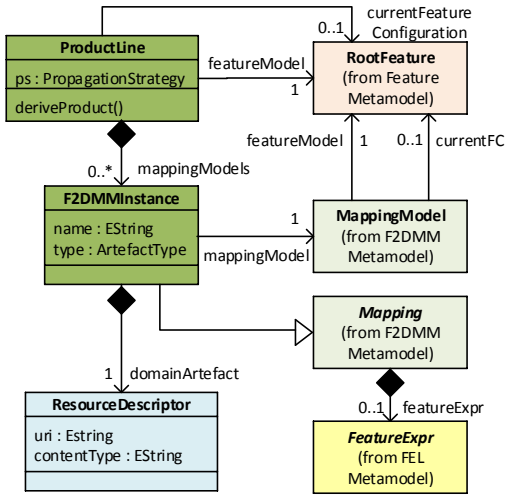


Figure 6. The FAMILE metamodel, which is designed to support heterogeneous software product lines.

A single resource mapping model, which refers to exactly one mapped EMF resource, is represented by F2DMMInstance. This meta-class defines a number of *resource-specific* parameters, such as the name and the *artefact type* (requirements, implementation, test, etc.). Please note that F2DMMInstance extends the abstract meta-class Mapping defined in the F2DMM metamodel, which manages variability by the use of feature expressions and the calculation of selection states [12]. Thus, variability on a coarse-grained level (i.e., on the level of resources) is enabled. The referenced MappingModel describes the mapping of the specific contents of a mapped resource, e.g., mapped EMF objects in the case of EMF model resources. Furthermore, a contained ResourceDescriptor element describes additional resource-specific parameters, being the relative URI of the mapped resource, as well as its *content type* (plain text, XML, EMF, etc.). The resource containing a multi-variant domain model is referenced by its URI.

Besides the possibility of annotating specific resources of the multi-variant domain model with feature expressions, the presented extension addresses the fact that in heterogeneous projects, *cross-resource links* occur frequently. For instance, in the example in Section VI, elements of an Ecore model are referenced by a corresponding GMF mapping model located in a different resource. Please note: from the viewpoint of FAMILE, the GMF mapping model is just an ordinary artefact that is also based on the Ecore metamodel. During product derivation, these links are detected and resolved automatically in order to meet the requirement of referential integrity across multiple resources.

B. Interpreting Non-Model Artifacts as EMF Instances

EMF and its metamodel Ecore are wide-spread in the Eclipse community, thus a large number of potential domain models is addressed by relying on EMF models as SPL artefacts. A (non-exhaustive) list comprises of course Ecore class diagrams, Eclipse UML models [20], Xtext [21] / EMFText [22] grammars and documents, GMF models [16], Acceleo source code generation templates [23], MWE2 Workflow files [24], Xtend specifications [25], domain-specific languages based on Ecore, and many more. Additionally, FAMILE has been applied successfully to Java source code as well. To this end, the MoDisco [26] framework is used, which allows to parse Java source code into a corresponding Java model instance (which is also based on Ecore). MoDisco may be also used to create EMF model instances out of XML files.

As explained before, the new framework considers all artefacts part of the platform as models. In this subsection, it is explained how particular resource types can be interpreted as EMF instances. The new FAMILE implementation allows for different extensions for specific heterogeneous resource types. For each resource type, different user interface extensions are provided, in order to allow the user to work at an adequate level of abstraction. So far, five resource types have been implemented.

- **XMI-serialized EMF models**. These are ordinary models, which may be mapped using the F2DMM editor, which represents the model as a tree.
- **Xtext models** may be mapped using their abstract syntax tree, since Xtext files implement EMF's `Resource` interface. Currently, it is not possible to add feature expressions to a text selection. It is planned to add this feature in future.
- **Java files**. Invisibly to the user, Java source code files are converted to EMF models using the MoDisco framework. The user may select a source code fragment and invoke the command *Annotate Java element*. Behind the scenes, the mapping is applied to the underlying MoDisco discovery model.
- **XML files**. Similarly to Java files, MoDisco offers a discoverer for XML. Currently, only the concrete syntax may be mapped using the standard F2DMM single resource mapping editor.
- **Unstructured text**. For text files that do not fit any of the categories above, the new FAMILE version includes a fall-back representation. Text files are represented as an instance of a simple text meta-model, which only consists of a sequence of text lines. This way, single text lines may be assigned with feature expressions using the F2DMM editor.

In the running example and in the screencast, the focus lies on two resource types, being XMI-serialized EMF models and Java files.

C. User Interface

The user interface has been extended to support heterogeneous software product lines. An additional editor manages the mapping for a set of resources rather than single-resource models, which are still covered by the existing F2DMM editor. In addition to the tasks listed in Section III, the extended FAMILE framework supports the following user interactions (see also example in Section VI):

1) **Adding heterogeneous product line support** An arbitrary Eclipse project containing any kind of resource (e.g., EMF models, source code and documentation) can be provided with the *FAMILE nature*, which adds heterogeneous product line support by automatically creating a FAMILE product line model.
2) **Definition of a global feature model** As soon as the FAMILE nature has been added, the feature model editor is opened automatically and can be used to provide the results of domain analysis. Of course, it is also possible to reuse an existing feature model. Once a new feature model has been created or an existing feature model has been selected, its contained features may be used in feature expressions annotating corresponding implementation fragments from the multi-variant domain model(s).
3) **Adding variability to resources** Initially, it is assumed that none of the project resources is subject to variability. In order to add variability to a specific resource, the *Add F2DMM Instance* command can be invoked. It will create a new mapping model for the selected resource and append it to the reference mappingModels of the ProductLine instance. Furthermore, global project parameters are transferred to the new F2DMM instance.

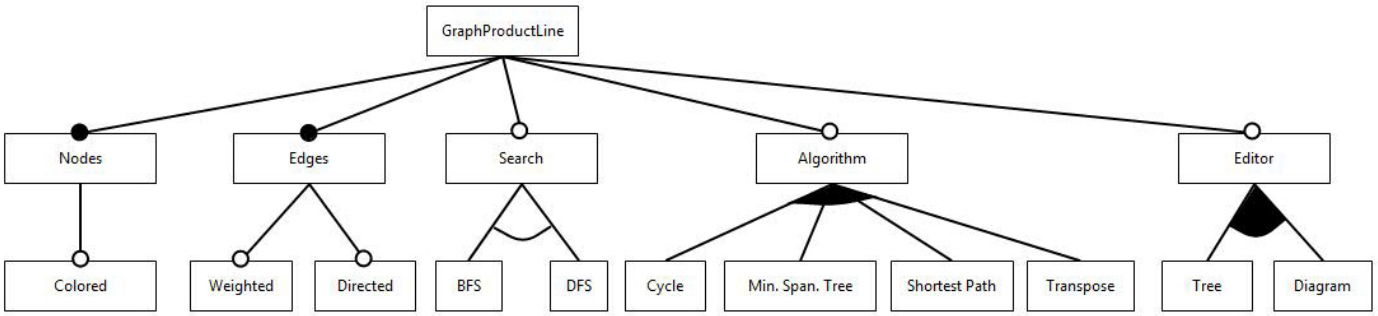


Figure 7. Feature diagram for the graph product line.

4) **Assigning feature expressions to resources** In many cases, variability is achieved at a rather coarse-grained level, having resources rather than objects implement features. The FAMILE editor supports this requirement by the possibility of assigning feature expressions to entire resources.
5) **Applying a feature configuration globally** The command *Set Feature Configuration* allows to change the current configuration, which will restrict the visible elements/resources in both the resource mapping and the resource set mapping editor to elements with a feature expression that satisfies the new configuration. This global project parameter is propagated to all existing F2DMM instances.
6) **Deriving a multi-resource product** After applying a specific feature configuration, a product can be exported. Invoking the *Derive Product* command will prompt the user for a name of the derived Eclipse project. As described above, single-resource product derivation will be applied to each mapping model covering a resource, keeping cross-resource links consistent. Resources that are not wrapped by any F2DMM instance or that are not annotated with FEL expressions will be copied without any further restriction.

VI. Example Revisited: Heterogeneous Product Line for Graph Metamodels and Corresponding Editors

In Section III, the development process that is commonly used in software product line engineering has been briefly sketched. In the following, it is demonstrated how to use FAMILE for model-driven product line engineering with this process, using the running example introduced in Section IV. In order to demonstrate the use of the heterogeneous extensions to FAMILE, the example product line is enriched with editors for the underlying graph data structure. In the domain engineering sub-process, the variability is captured in a variability model and a platform is established, which consists of not only the graph metamodel, but also of resources that control the aspects of representation as well as behavioral components – i.e., graph algorithms – which are described at the level of Java source code.

This section is organized as follows. The (heterogeneous) platform and the variability model are introduced in Subsection VI-A. In Subsection VI-B, the mapping between platform and variability model is described, with a focus on heterogeneous resources. Subsection VI-C refers to the use of two existing FAMILE concepts, being *alternative mappings* and *propagation strategies*. In Subsection VI-D, the created mapping is investigated with a focus on the interplay between heterogeneous artefacts, i.e., different types of links among them. The transition from domain engineering to application engineering – i.e., product derivation – is subject of Subsection VI-E. Subsection VI-F gives an outlook for a more fine-grained specification of variability among the platform.

A. Platform and Variability Model

The platform of the product line contains the following types of artefacts:

- **Ecore Model:** An Ecore model is used to describe the static structure of the product line for graph libraries. EMF allows to generate Java code for the model as well as code for a tree editor from the Ecore specification.
- **Java Code:** As model-driven development using EMF only allows to model the static structure of a software system, it is necessary to supply the corresponding method bodies using hand-written Java code. These method bodies also contain variability, which needs to be managed by FAMILE.
- **GMF Models:** The (optional) graphical editors for the graph product line have been developed using the Eclipse Graphical Modeling Framework (GMF) [16].

The variability of the graph product line is captured in a feature model. The corresponding feature diagram is depicted in Figure 7. A graph consists of *Nodes* and *Edges*. The graphical notation uses filled dots for mandatory and unfilled ones for optional features. Nodes may be *Colored* while edges may be *Weighted* and/or *Directed*. Optional components of the graph product line are the *Search* strategy (e.g., depth-first search or breadth-first search), different *Algorithm*s and *Editor*s. Child elements of feature groups are depicted with arcs. Two different types of group relationships are possible: "inclusive or" (filled arc) and "exclusive or" (unfilled arc) of child elements. While the different search strategies are mutually exclusive, the algorithms and the editor children may be selected in arbitrary combinations. Please note that there are also dependencies between features, which cannot

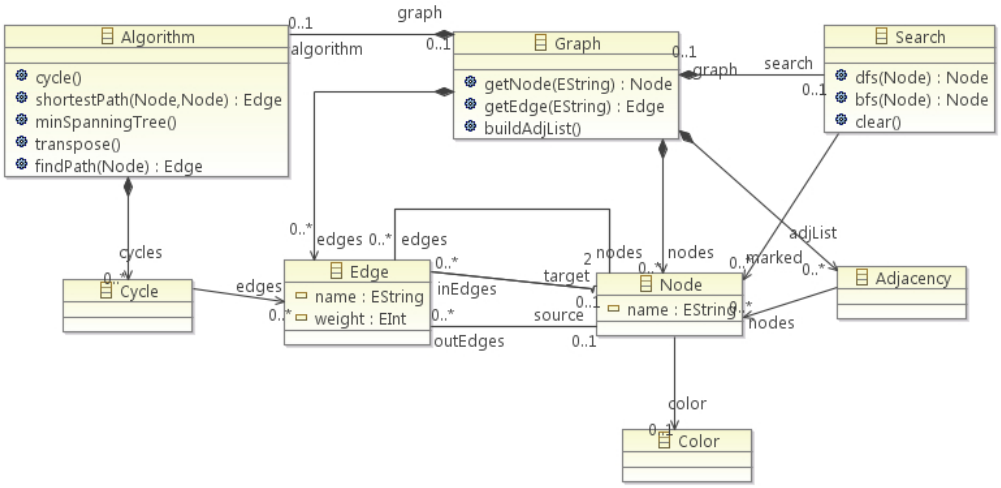


Figure 8. Ecore model for the graph product line.

be displayed in the feature diagram. Features may also define cross-tree constraints indicating feature inclusion or exclusion once a certain feature is selected. E.g., the algorithm to detect cycles in the graph requires *Directed* edges. Furthermore, the calculation of a *Shortest Path* requires *Weighted* edges.

Figure 8 depicts the multi-variant domain model of the graph product line. Following the model-driven approach, an object-oriented decomposition of the underlying data structure is applied: A Graph contains Nodes and Edges. Furthermore, it may contain a Search strategy and Algorithms operating on the graph data structure. For performance reasons, the data structure may be converted into an Adjacency list, to speed up certain algorithms. As the model depicted in Figure 8 is the superimposition of all variants, the relation between nodes and edges is expressed in multiple ways: (1) In case of undirected graphs, an edge is used to simply connect two nodes, expressed by the reference nodes. (2) Directed graphs on the other hand demand for a distinction of the corresponding start and end nodes of an edge. This fact is expressed by two single-valued references named source and target.

As stated above, Ecore only allows for structural modeling, i.e., it does not provide support to model method bodies. Thus, the standard EMF development process [3] demands for a manual specification of an EOperation's body by completing the generated source code. In the example, hand-written Java source code for all operations contained in the class diagram shown in Figure 8 has been supplied. A small cut-out of a method implementation for the class Search is shown in Figure 9. In the corresponding Ecore model (cf. Figure 8), the Search class defines three EOperations. While the EMF code generation only creates Java code for the method head, the body implementation depicted in Figure 9 was supplied manually. In this case, the method implementation also contains variability as the corresponding references between nodes and edges are different depending on the presence or absence of the feature *Directed* in the current feature configuration. Please note that the level of granularity supported by FAMILE's variability annotations is arbitrary, ranging from single Java fragments, over statements, blocks, methods or even classes and packages.

```
246     /**
247      * <!-- begin-user-doc -->
248      * <!-- end-user-doc -->
249      * @generated NOT
250      */
251     public EList<Node> dfs(Node node) {
252         if (getMarked().contains(node))
253             return getMarked();
254         getMarked().add(node);
255
256         // undirected graphs
257         for (Edge e : node.getEdges()) {
258             for (Node n : e.getNodes()) {
259                 if (n != node)
260                     dfs(n);
261             }
262         }
263
264         // directed graphs
265         for (Edge e : node.getOutEdges()) {
266             dfs(e.getTarget());
267         }
268
269         return getMarked();
270     }
```

Figure 9. Example for method bodies written in Java.

The product line for graphs also allows for different types of editors, which may be used to manipulate the graph data structure. As shown in the corresponding feature model in Figure 7, valid product configurations may either have no editor at all, a tree editor, a graphical editor or they may even contain both types of editor. While the tree editor may be automatically generated from the Ecore model, a graphical editor requires additional information. The Graphical Modeling Framework (GMF) [16], which is also part of the Eclipse Modeling Platform, allows for a creation of graphical editors in a model-driven way. Generating graphical editors with GMF requires the definition of three additional models:

1) **GMFGraph (Graphical Definition Model)** GMF uses a GMFGraph model to define the graphical representation of the concrete syntax. In case of the example, the visual appearance of nodes and edges of the graph is defined, by specifying the respective

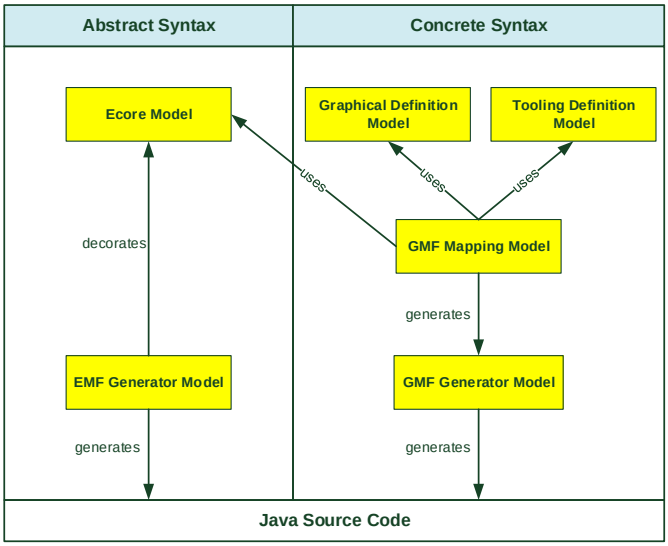


Figure 10. Models involved in the GMF development process.

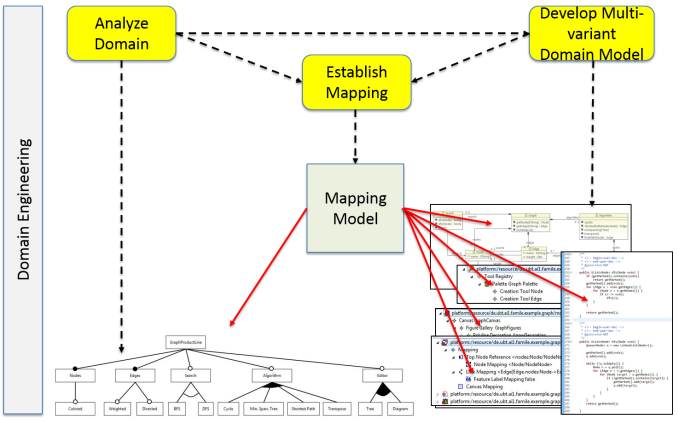


Figure 11. Domain Engineering Process in the graph product line example.

shapes.

2) **GMFTool (Tooling Definition Model)** Every graphical editor in Eclipse, which is based on the Graphical Editing Framework (GEF), uses a so called palette to drag new diagram elements to the drawing canvas. As GMF is a model-driven extension to GEF, it follows this paradigm. The GMFTooling definition model is used to specify the contents of the editor's tool palette.
3) **GMFMap (GMF Mapping Model)** The models described above are combined in the GMF mapping model. In this model, a relation between abstract syntax (Ecore), the graphical notation (GMFGraph) and the tooling definition model (GMFTool) is established. The GMF Mapping Model is then automatically transformed into a generator model from which the Eclipse plugin for the graphical editor is generated. Please note that the GMF mapping model is the central part of the Graphical Modeling Framework. It has nothing in common with the F2DMM mapping model, which is the core of the FAMILE tool chain. From the viewpoint of FAMILE, all the models that have been described here are just ordinary artefacts that may contain variability.

Figure 10 depicts the different models involved in the GMF development process. All models are instances of Ecore-based metamodels, and can thus be used easily with the FAMILE tool chain. The abstract syntax of a GMF-based editor is defined by an Ecore model (in this case, the superimposed graph metamodel shown in Figure 8), while the editor providing the concrete (graphical) syntax is defined by a graphical definition model, a tooling definition model and a GMF mapping model.

The EMF generator model is used to generate Java source code for the abstract syntax while the GMF generator model is responsible for generating the diagram editor's source code. Please note that the screencast complementing this paper does not cover the definition of the models mentioned below as it just focuses on how to use FAMILE with these types of artefacts.

B. Mapping Heterogeneous Artefacts

Figure 11 depicts the domain engineering phase in model-driven software product lines developed with FAMILE. First, the variability has to be captured in a feature model. The features are then implemented using the appropriate modeling languages. After that, a mapping between features and their corresponding implementation fragments has to be established. In the following subsection, the necessary steps are described from the tool perspective.

In order to use FAMILE for a (heterogeneous) project, the FAMILE project nature has to be assigned. As a result, an empty feature model and a FAMILE model are created within the project. Variability modeling, i.e., capturing the commonalities and differences of the products in the product line, is performed during the domain analysis step. The result of this development task is a feature model. In the example, the feature model shown in Figure 7 is applied to the entire product line as a global project parameter. In order to map features to corresponding implementation fragments, F2DMM mapping models have to be created for each domain model. In the example, five F2DMM instances have been defined, one for each EMF/GMF resource mentioned above and one for the Java source code — the underlying MoDisco AST representation consists of a single EMF resource although the source code is distributed over multiple packages and compilation units in the physical file system. Please note that for the EMF Generator model and the GMF Generator model (cf. Figure 10) no F2DMM instances have been defined, as they do not contain variability in this example. Furthermore, those models have been automatically derived from the Ecore model and the GMF mapping model and only contain information for the code generators. Thus, they can easily be created again after product derivation.

Since the graph metamodel as well as all GMF artefacts are ordinary EMF models, their mapping is done in a straightforward way using the single resource F2DMM mapping editor (see Section IV). For mapping the source code, the new concrete syntax connector of the FAMILE extension may be used.

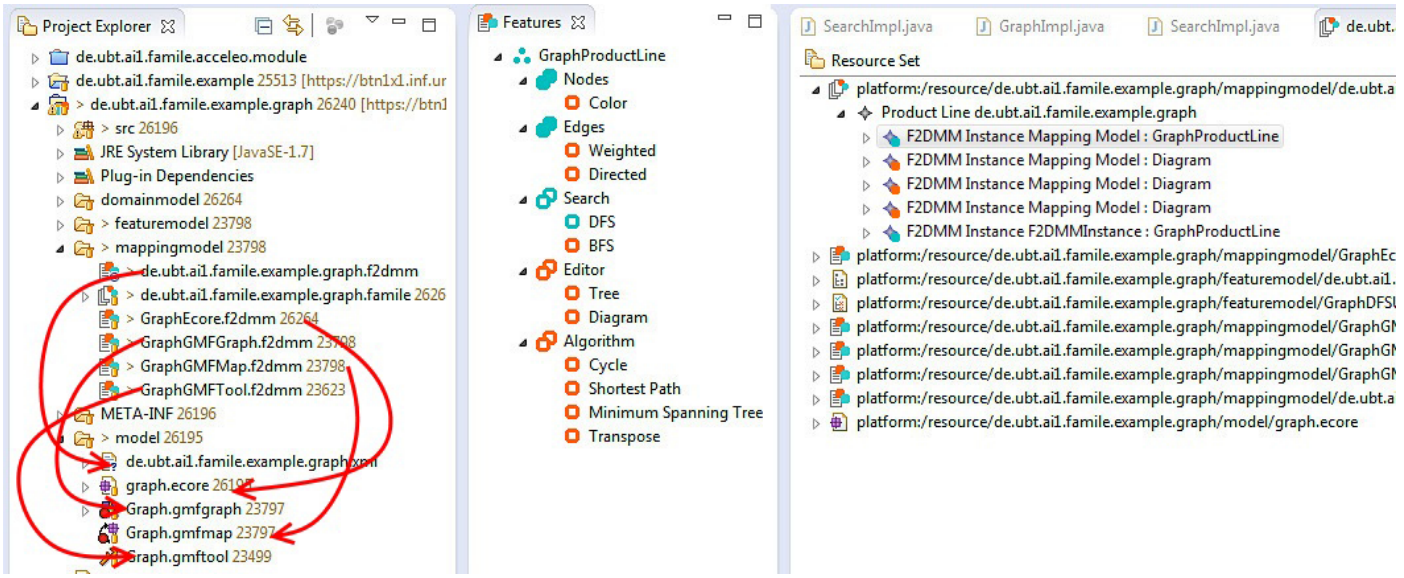


Figure 12. Screenshot of the FAMILE resource set mapping editor. The left pane shows the feature model and feature configuration. In the main pane, the mapping for contents of the resource set are shown.

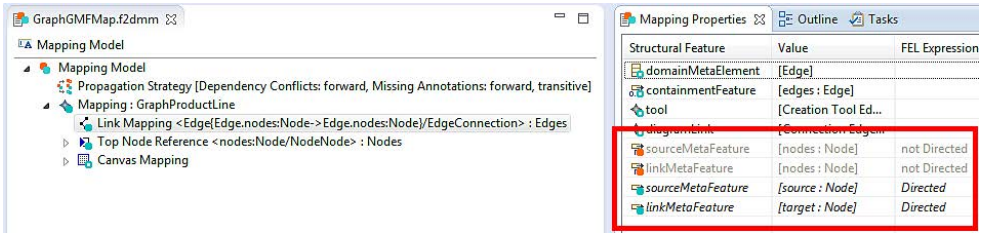


Figure 13. Usage of alternative mappings. The red box depicts where elements of the multi-variant domain model have been virtually extended by alternative mapping values (in italics).

It allows to assign feature expressions to parts of the hand-written method bodies directly in the textual representation of the Java source code. The body of the method `dfs` shown in Figure 9 is annotated as follows (see also screencast) in order to provide an optimized depth-first search for directed graphs. First, the user selects lines `257` until `262` in the editor. Next, he/she invokes the command `Annotate Java element` from the context menu. The entered feature expression "Directed" will be assigned to the corresponding AST element in the MoDisco model in the background. Similarly, the `for`-loop consisting of lines `265` until `267` is restricted by the feature expression "not Directed". As a consequence, derived products will only contain one of the two loops to calculate the successors for a depth-first search, but never both of them.

Figure 12 depicts the state of the example project after corresponding F2DMM instances have been created for the models mentioned above. The red arrows in the left part of the figure indicate which domain model resource the corresponding F2DMM models refer to. As one can see, mapped resources may also be annotated with feature expressions. For the example, a feature called *Editor* has been introduced in order to make the visualization (tree editor vs. diagram editor) of the graph variable. In case the feature *Diagram* is deselected in a feature configuration, it is obvious that the resulting product must not contain the GMF models. As a consequence, the respective F2DMM instances are annotated with the feature expression Diagram, as shown in Figure 12.

C. Usage of Alternative Mappings and Propagation Strategies

Figure 3 has already shown the content of the F2DMM mapping model for the Ecore model, which is used to define the abstract syntax of the graph model. Analogously, F2DMM instances for the other required artefacts (GMFGraph, GMFTool, GMFMap and Java code) are created. Each model file contains a superimposition of all possible variants. Common approaches using negative variability suffer from restrictions imposed by the used domain metamodels, which usually do not provide adequate support for variability. FAMILE mitigates this restriction by offering the advanced concept of *alternative mappings*. In the example, alternative mappings are used in the Link mapping in the GMFMap model (cf. Figure 13). In case of an undirected graph, the corresponding graphical editor should just connect two nodes by a solid line. To this end, the underlying semantic model (i.e., the Ecore class model) provides a reference nodes in the class Edge. In contrast, if the feature *Directed* edges is selected, the graphical editor should indicate the direction of the edge connecting two nodes by using an arrow as a target decorator. Furthermore, the semantic model does no longer contain a reference nodes, but instead two single-valued references source and target,

which are used to store the corresponding nodes connected by the edge. In GMF, a link mapping requires to specify the corresponding EReferences used as the link's source and target. While in the first case, both source and target features in the GMFMap file are set to the EReference nodes, the latter case requires those features to point at the corresponding source and target EReferences.

In this example, FAMILE's alternative mapping capabilities are necessary because the GMF mapping model uses a single-valued EReference to store the sourceMetaFeature and linkMetaFeature features. In case of undirected edges, the nodes Reference defined in the Ecore model of the graph product line is used. However, in case of directed edges, a distinction between source and target nodes is required. To this end, the Ecore model provides corresponding source and target EReferences in the class Edge (cf. Figure 8), which have to be used in the GMFMap model instead in case the feature *Directed* is chosen. Please note that the reason why in this example, alternative mappings are necessary in the GMF mapping model but not in the semantic model is the fact that the references `nodes`, `source`, `target` can be defined simultaneously in the Ecore model. The GMF mapping model, however, requires the applied occurrence of exactly one semantic model element here, which cannot be realized by a single multi-variant model. Figure 13 depicts how this has been solved using FAMILE's alternative mappings [12], which can virtually extend the multi-variant model and thus mitigate the limited variability of the used domain metamodels.

It is aimed to keep the annotation effort small for the users of the tool chain. This can be achieved by so called propagation strategies, which avoid the necessity of repeated feature annotations. In the graph product line example, the propagation strategy *forward* is used throughout all mapped resources. As its name says, this strategy propagates selection states of mapped elements *along the direction of dependency*. For instance, each EMF object has an existential dependency to its `eContainer`. As a consequence, in case the presence of an element is restricted by a specific feature expression, this restriction also holds for all contained elements, making repeated annotations unnecessary. For additional details on propagation strategies, the reader is referred to [13].

D. Interplay Between Different Model Types

Through the use of two external frameworks, namely GMF and MoDisco, the resulting example product line is highly heterogeneous. Figure 15 sketches the interplay between different resource types, being the domain model, the GMF models, the generated Java source code and its internal MoDisco representation, which is invisible to the end user. Between different resource types, seven external link types occur, two of which are *conceptual*, i.e., they do not occur explicitly as EMF links.

- **GMF Mapping links** emerge from the GMF mapping model and reference the domain model, the tooling model, and the graphical definition model. These links are created by GMF.
- **Conceptual CS/AS links** virtually link an abstract syntax tree element to its corresponding concrete syntax fragment in the Java source code. This link is restored automatically in case the user selects a concrete syntax fragment and invokes the command *Annotate FEL Expression*.
- **Feature expression links**: Within the feature expression of a mapping, elements of the feature model are referenced. These links are created automatically when feature expressions are specified.
- **Conceptual links between CS and feature expression**: From the user's perspective, source code elements are mapped in concrete syntax. This causes a conceptual dependency between source code fragments and features.
- **Mapping links**: Each element of the mapping model references exactly one element of the multi-variant domain model. These links are created automatically upon creation of a resource mapping model.
- **Cross-Resource Mapping links**: Among different domain models, cross-resource links may occur (see the GMF mapping model). In order to adequately connect these to the variability model, a link between the corresponding resource mappings is established. These links are created automatically during mapping model creation.
- **Resource Mapping links**. The superordinate resource set mapping model references one F2DMM instance per mapped model resource. Links of this kind are created by the user through the *Add F2DMM Instance* command.

By automating sub-tasks such as the synchronization between domain and mapping model, and the discovery of the Java source code, the larger part of the complex relationships shown in Figure 15 is not exposed to the user at all, but managed automatically "behind the curtains".

E. Product Derivation

After the domain engineering phase has been completed, the platform may be used to derive specific products from the product line in the application engineering step (cf. Figure

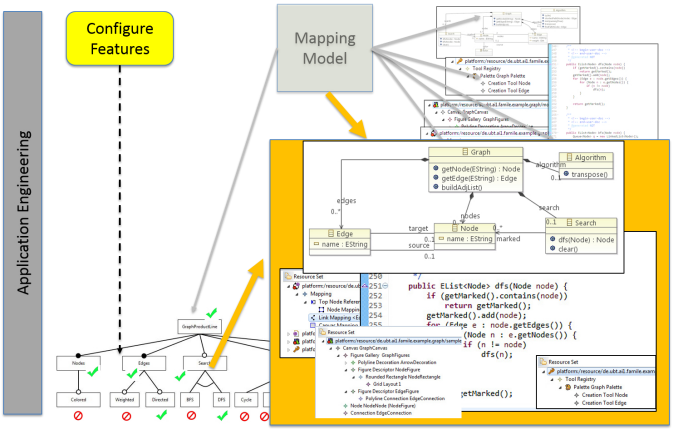


Figure 14. Application Engineering Process in the graph product line example.

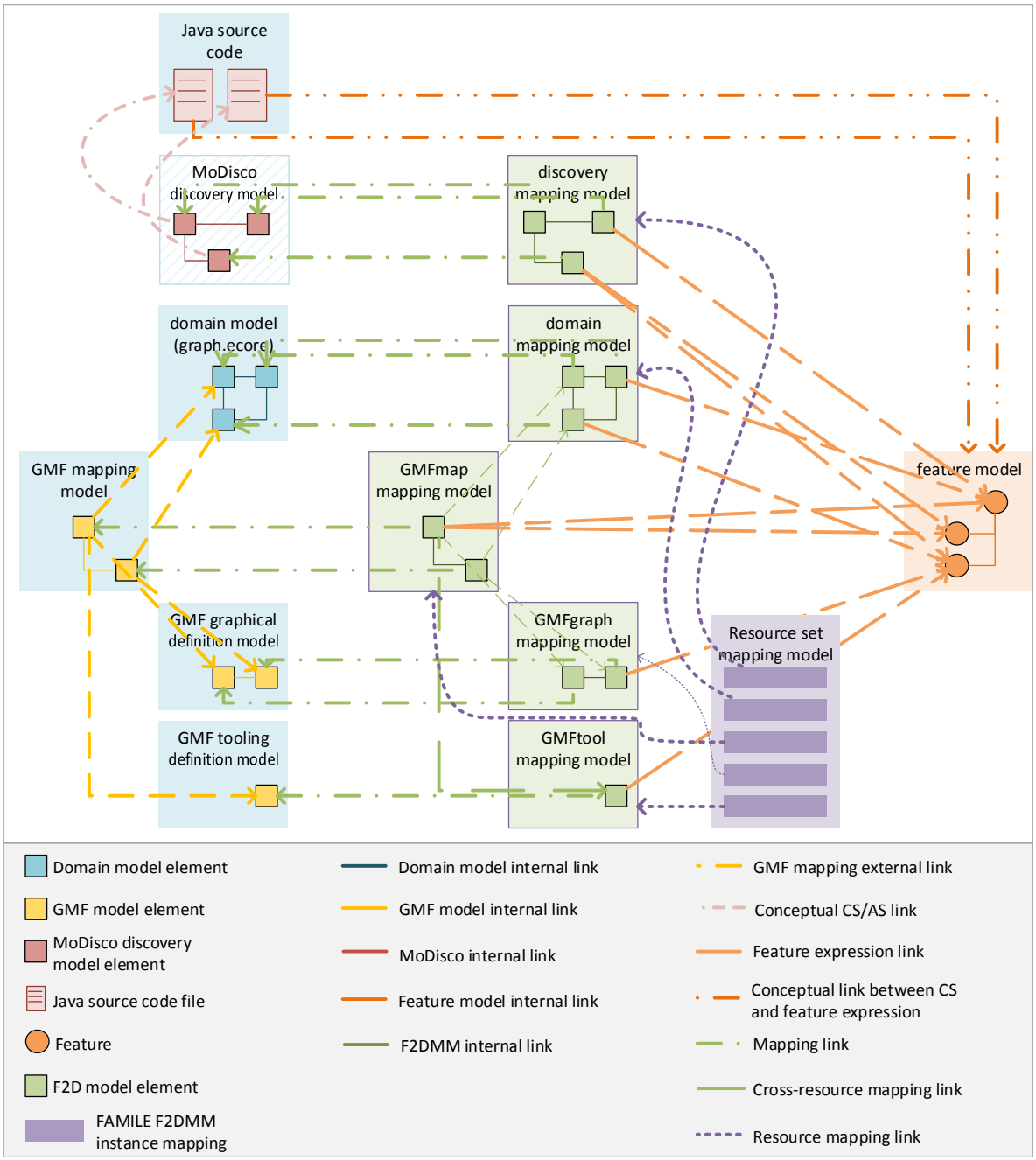


Figure 15. Interplay between resources of different types, which are created in subsequent steps of the *domain engineering* phase.

14). In the presented approach, application engineering is reduced to a simple configuration task. The user only has to specify an appropriate feature configuration, by selecting and deselecting corresponding features in the feature model, while all constraints (e.g., parent-child relationships, requires/excludes relationships) must be satisfied. The subsequent product derivation step is a fully automatic process.

In the example, a derived Eclipse project is created, which contains the required model files. Sample feature configurations are provided, which allow for a fully automatic generation of four Eclipse plugin projects, which differ from each other as follows (cf. Figure 16):

(FC1) An EMF tree editor for undirected, unweighted, uncolored graphs, traversed by depth-first search. No additional algorithms are offered.

(FC2) A GMF graphical editor for undirected, unweighted, uncolored graphs, traversed by depth-first search. No additional algorithms are offered.

(FC3) A GMF graphical editor for directed, unweighted, uncolored graphs, traversed by depth-first search. Deployed algorithms include cycle detection, as well as calculation of a minimum spanning tree and the transpose.

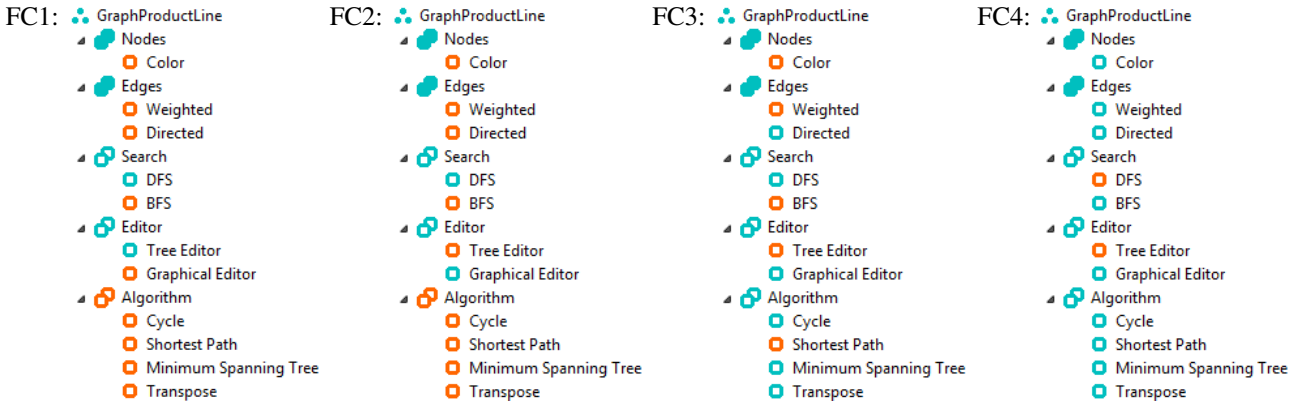


Figure 16. Four example feature configurations for instances of the graph product line.

(FC4) A GMF graphical editor for directed, weighted, colored graphs, traversed by breadth-first search. All available graph algorithms are deployed.

Of course, this set of feature configurations does not contain all possible combinations of features and it may be extended arbitrarily based on the features and constraints defined in the feature model.

F. Outlook: Increasing the Heterogeneity of the Project

The example described in this section has been conducted using a variety of different resources as artefacts. All models involved in the GMF development process, i.e., the Ecore domain model, the Graphical Definition Model, the Tooling Definition Model, as well as the GMF Mapping Model, are instances of different Ecore-based metamodels. Manual adaptations to the Java source code are managed with the help of the MoDisco framework, letting the user operate on concrete textual syntax. In the current state of the project, these models constitute the adequate level of abstraction for variability management. However, it might become necessary to define additional F2DMM mapping models for additional non-EMF resources, for different reasons:

- The file `plugin.properties` in the Eclipse project contains language-specific UI string constants, each declared in a separate text line. Currently, the generated Editor displays UI elements in English. However, if support for different languages is desired, one may add an additional F2DMM mapping model for the properties file, and corresponding features for each additional language to the feature model. The mapping may be adequately managed by means of a per-line mapping, using the "fall-back" EMF representation for plain text files (see Section V-B).
- The file `plugin.xml` defines plug-in extensions used to integrate the generated editor with the Eclipse platform. By adding an F2DMM mapping model and corresponding features, variability may be added to the plugin's runtime configuration, i.e., in order to make the editor's icon, label, or file extension depend on specific feature configurations. Assuming that no EMF-compatible metamodel for Eclipse plugin files is defined, the MoDisco based representation for XML files (see Section V-B) may be used.

VII. Related Work

Many different tools and approaches have been published in the last few years, which address (model-driven) software product line development. Due to space restrictions, the focus of this comparison lies on support for heterogeneous software projects, using the definition of heterogeneity given in the introduction. Other comparisons of FAMILE and related approaches can be found in [12] and [13].

The tool *fmp2rsm* [27] combines FeaturePlugin [28] with IBM's Rational Software Modeler (RSM), a UML-based modeling tool. The connection of features and domain model elements is realized by embedding the mapping information into the domain model using stereotypes (each feature is represented by its own stereotype), which requires manual extensions to the domain model. While fmp2rsm is limited to the support of RSM models, the approach presented in this paper provides a greater flexibility since the only restriction is that the domain model needs to be Ecore based. Furthermore, the extensions presented in this paper allow to use several domain metamodels within one software product line project.

FeatureMapper [10] is a tool that allows for the mapping of features to Ecore based domain models. Like FAMILE, it follows a very general approach permitting arbitrary Ecore models as domain models. FeatureMapper only allows to map a single (self-contained) domain model, while the work presented in this paper allows to use FAMILE also for software product lines whose multi-variant domain model is composed of artefacts distributed over different resources. Furthermore, the artefacts may be instances of different metamodels.

VML* [8] is a family of languages for variability management in software product lines. It addresses the ability to explicitly express the relationship between feature models and other artefacts of the product line. It can handle any domain model as long as a corresponding VML language exists for it. VML* supports both positive and negative variability as well as any combination thereof, since every action is a small transformation on the core model. As a consequence, the order in which model transformations are executed during product derivation becomes important. So far, VML* is designed to

work with text files, provided that a corresponding VML language exists for it (i.e., a grammar has to be specified). Theoretically, VML languages could be written that work with XMI serializations of the respective models in the example presented in this paper, whereas FAMILE provides generic support for model-driven software development based on Ecore compliant models. In other words, VML* and FAMILE provide similar support for heterogeneous projects, but they operate on different "technological spaces". As a consequence, the example provided in Section VI cannot be realized with VML* easily. In fact, significant effort would be required to create VML languages for the different models involved in the graph product line example as presented here.

MATA [9] is another language that also allows to develop model-driven product lines with UML. It is based on positive variability, which means that, around a common core specified in UML, variant models described in the MATA language are composed to a product specific UML model. Graph transformations based on AGG [29] are used to compose the common core with the single MATA specifications. While MATA is limited to UML, the approach presented in this paper provides support for any Ecore based model and furthermore allows the combination of different domain metamodels within one product line project.

CIDE [11] is a tool for source-code based approaches. It provides a product specific view on the source code, where all source code fragments not part of the chosen configuration are omitted. The approach is similar to *#ifdef*-preprocessors known from the C programming language [30]. The difference is that it abstracts from plain text files and works on the abstract syntax tree of the target language instead. In its current state, CIDE provides support for a wide range of different programming languages. Unfortunately, it cannot be used for model-driven development. In contrast, FAMILE provides full-fledged support for model-driven development based on Ecore models. Furthermore, it may also deal with regular Java source code by using the MoDisco [26] framework.

Bühne et al. [31] and Dhungana et al. [32] present approaches for heterogeneous variability modeling, i.e., managing commonalities and differences across *multi product lines*. Dhungana et al. aim at unifying multi product lines, which rely on different tools and formalisms for modeling variability. Web services are used for a prototypical implementation. In contrast to the approach presented here, in both approaches, the term 'heterogeneity' concerns different variability models rather than the product space. While Bühne et al. and Dhungana et al. only address variability modeling, the approach presented in this paper covers a larger part of the software life-cycle. Furthermore, FAMILE does not only allow for variability modeling, but also for mapping the variability information to heterogeneous implementation artefacts.

VIII. Conclusion and Future Work

In this paper, requirements, concepts and limitations with respect to tool support for heterogeneous model-driven software product lines have been discussed. The approach presented in this paper closes a significant gap in the tool support for model-driven development of software product lines, whose artefacts are heterogeneous in terms of the used metamodels as well as in containing artefacts like source code or configuration files or XML documents. As a proof of concept, an implementation of an extension to the FAMILE tool chain was shown. A concrete example has been given, demonstrating the benefits of the presented approach on a concrete product line for graphs.

Usually, (model-driven) software projects do not only consist of one single model resource. In contrast, different models and metamodels as well as non-model artefacts are involved. FAMILE is able to map features to model fragments in such heterogeneous projects and also to derive consistent products. Besides the aforementioned heterogeneous support, FAMILE advances the state of the art by allowing to flexibly change the granularity of the mapping between features and the product space (project-wide or resource-wide scope). Furthermore, the tool chain also allows for the usage in model-driven projects, where parts of the software are still realized with manually written Java source code. Of course, FAMILE may be used in regular (non model-driven) Java projects as well. The main challenges of heterogeneous SPLE tool support are (a) to cope with different levels of abstractions (models and source code / plain text files) as well as (b) different forms of representation, (c) to ensure that links between different resources are kept consistent, (d) to adequately handle conceptual links between artefacts of different types, e.g., between model elements and source code fragments, and (e) to provide a uniform variability mechanism with respect to all project resources.

The approach presented here comes with the assumption that each resource type may be expressed by an EMF model; the new version of FAMILE provides adequate mapping constructs in order to support entire Eclipse projects. Furthermore, the presented solution to heterogeneous SPLE tooling is to divide a heterogeneous software project into a set of single-resource mapping models, for which adequate MDPLE support is already available. Links between different models are kept consistent during product derivation. Extensions to the user interface ease the integration of new artefacts into heterogeneous product lines as well as modifications to existing mappings. Non-model resources such as Java or XML files are automatically interpreted as EMF models, using the MoDisco framework under the hood. Furthermore, a fallback metamodel for text files is provided, which also allows to map features to those kinds of artefacts at a lower level of abstraction. A demonstration of the presented approach was given by applying the heterogeneous FAMILE tool chain to a product line for graph metamodels, including modifications of the generated source code, and editors. The resulting heterogeneous product line manages an entire Eclipse plug-in project.

Current and future work addresses a case study carried out in the field of robotics [33][34]. Although first results produced by the old (homogeneous) version of the FAMILE tool chain are very promising, it is expected that a significant gain in productivity is achieved by exploiting the new, heterogeneous approach. In this case study, the platform of the product line consists of language grammar files, code generation template files and C++ source code files.

Acknowledgments

The authors want to thank Bernhard Westfechtel for his valuable comments on the draft of this paper.

REFERENCES

[1] T. Buchmann and F. Schwägerl, "A model-driven approach to the development of heterogeneous software product lines," in *Proceedings of the Ninth International Conference on Software Engineering Advances (ICSEA 2014)*, H. Mannaert, L. Lavazza, R. Oberhauser, M. Kajko-Mattsson, and M. Gebhart, Eds. Nice, France: IARIA, 2014, pp. 300–308.

[2] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[3] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[4] OMG, *Meta Object Facility (MOF) Core*, formal/2011-08-07 ed., Object Management Group, Needham, MA, Aug. 2011.

[5] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA, 2001.

[6] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer Verlag, 2005.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[8] S. Zschaler, P. Sánchez, J. Santos, M. Alférez, A. Rashid, L. Fuentes, A. Moreira, J. Araújo, and U. Kulesza, "VML* - A Family of Languages for Variability Management in Software Product Lines," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, M. van den Brand, D. Gaevic, and J. Gray, Eds. Denver, CO, USA: Springer Berlin / Heidelberg, 2010, vol. 5969, pp. 82–102.

[9] J. Whittle, P. Jayaraman, A. Elkhodary, A. Moreira, and J. Arajo, "MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation," in *Transactions on Aspect-Oriented Software Development VI*, ser. Lecture Notes in Computer Science, S. Katz, H. Ossher, R. France, and J.-M. Jzquel, Eds. Springer Berlin / Heidelberg, 2009, vol. 5560, pp. 191–237.

[10] F. Heidenreich, J. Kopcsek, and C. Wende, "FeatureMapper: Mapping features to models," in *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany, May 2008, pp. 943–944.

[11] C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. S. Batory, "Guaranteeing syntactic correctness for all product line variants: A language-independent approach," in *TOOLS (47)*, ser. Lecture Notes in Business Information Processing, M. Oriol and B. Meyer, Eds., vol. 33. Springer, 2009, pp. 175–194.

[12] T. Buchmann and F. Schwägerl, "FAMILE: tool support for evolving model-driven product lines," in *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications*, ser. CEUR WS, H. Störrle, G. Botterweck, M. Bourdells, D. Kolovos, R. Paige, E. Roubtsova, J. Rubin, and J.-P. Tolvanen, Eds. Building 321, DK-2800 Kongens Lyngby: Technical University of Denmark (DTU), Jul. 2012, pp. 59–62.

[13] T. Buchmann and F. Schwägerl, "Ensuring well-formedness of configured domain models in model-driven product lines based on negative variability," in *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, ser. FOSD 2012. New York, NY, USA: ACM, 2012, pp. 37–44.

[14] OMG, *UML Superstructure*, formal/2011-08-06 ed., Object Management Group, Needham, MA, Aug. 2011.

[15] T. Buchmann, A. Dotor, and B. Westfechtel, "Mod2-scm: A model-driven product line for software configuration management systems," *Information and Software Technology*, 2012, http://dx.doi.org/10.1016/j.infsof.2012.07.010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2012.07.010

[16] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[17] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[18] F. Heidenreich, "Towards systematic ensuring well-formedness of software product lines," in *Proceedings of the 1st Workshop on Feature-Oriented Software Development*. Denver, CO, USA: ACM, Oct. 2009, pp. 69–74.

[19] R. E. Lopez-Herrejon and D. S. Batory, "A standard problem for evaluating product-line methodologies," in *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, ser. GCSE '01. London, UK, UK: Springer-Verlag, 2001, pp. 10–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=645418.652082

[20] "Eclipse UML2 Project," http://www.eclipse.org/modeling/mdt/?project=uml2, accessed: 2014-07-15.

[21] "Xtext project," http://www.eclipse.org/Xtext, accessed: 2014-07-15.

[22] "EMFText Project," http://www.emftext.org, accessed: 2014-07-15.

[23] "Acceleo project," http://www.eclipse.org/acceleo, accessed: 2014-07-15.

[24] "MWE2 Project," http://www.eclipse.org/modeling/emft/?project=mwe, accessed: 2014-07-15.

[25] "Xtend project," http://www.eclipse.org/xtend, accessed: 2014-07-15.

[26] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering," in *Proceedings of the IEEE/ACM International Conference on Automated software engineering (ASE 2010)*, Antwerp, Belgium, 2010, pp. 173–174.

[27] "fmp2rsm project," http://gsd.uwaterloo.ca/fmp2rsm, accessed: 2014-07-15.

[28] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature modeling plug-in for Eclipse," in *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (eclipse'04)*, New York, NY, 2004, pp. 67–72.

[29] G. Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software," in *Applications of Graph Transformations with Industrial Relevance*, ser. Lecture Notes in Computer Science, J. Pfaltz, M. Nagl, and B. Böhlen, Eds. Charlottesville, VA, USA: Springer Berlin / Heidelberg, 2004, vol. 3062, pp. 446–453.

[30] B. W. Kernighan, *The C Programming Language*, 2nd ed., D. M. Ritchie, Ed. Prentice Hall Professional Technical Reference, 1988.

[31] S. Bühne, K. Lauenroth, and K. Pohl, "Modelling requirements variability across product lines," in *RE*. IEEE Computer Society, 2005, pp. 41–52.

[32] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. A. Galindo, "Configuration of multi product lines by bridging heterogeneous variability modeling approaches," in *SPLC*, E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, Eds. IEEE, 2011, pp. 120–129.

[33] J. Baumgartl, T. Buchmann, D. Henrich, and B. Westfechtel, "Towards easy robot programming: Using dsls, code generators and software product lines," in *Proceedings of the 8th International Conference on Software Paradigm Trends (ICSOFT 2013)*, J. Cordeiro, D. Marca, and M. van Sinderen, Eds. ScitePress, Jul. 2013, pp. 548–554.

[34] T. Buchmann, J. Baumgartl, D. Henrich, and B. Westfechtel, "Towards a domain-specific language for pick-and-place applications," in *Proceedings of the Fourth International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2013).*, U. P. S. Christian Schlegel and S. Stinckwich, Eds. arXiv.org, 2013. [Online]. Available: http://arxiv.org/abs/1401.1376

A Modular Architecture of an Interactive Simulation and Training Environment for Advanced Driver Assistance Systems

Kareem Abdelgawad, Bassem Hassan, Jan Berssenbrügge, Jörg Stöcklein, and Michael Grafe
Heinz Nixdorf Institute
University of Paderborn, Germany
{Kareem.Abdelgawad, Bassem.Hassan, Jan.Berssenbruegge, Joerg.Stoecklein, Michael.Grafe}@hni.upb.de

Abstract - **Advanced Driver Assistance Systems (ADAS) are mechatronic vehicle systems that collaborate with the driver to improve road safety and increase driving comfort. Apart from all technical challenges regarding control algorithms and sensor quality, customer acceptance of ADAS is an important concern to automobile manufacturers. Simulating ADAS and demonstrating their benefits to customers in real traffic environments are impractical and leads to significant efforts and costs. This paper presents a modular architecture of a driving simulation environment for ADAS demonstration using driving simulators. The structure of the driving simulation environment is discussed. Special focus is given to the embedded framework for ADAS virtual prototyping and demonstration. This framework is built in a flexible form that ensures system scalability. That is, new ADAS prototypes can be designed and added almost without significant input-output interface adjustments. Furthermore, different ADAS can be integrated together to implement more advanced capabilities, such as autonomous driving. The framework is composed of modular functional units, which enclose real-time capable simulation models developed with MATLAB/Simulink. The design of the functional units and the input-output relationships are presented. Prototypes for Emergency Brake Assist and Emergency steer Assist are presented as examples of innovative ADAS that can be demonstrated using the developed simulation environment.**

Keywords - Advanced Driver Assistance Systems (ADAS); Driving simulators; Virtual prototyping; MATLAB/Simulink.

I. INTRODUCTION

Driving is one of the most popular daily activities that people perform. Nevertheless, it is a complex and relatively dangerous activity. Drivers have to concentrate on many tasks at the same time. Improving road safety standards is one of the main concerns in the automotive industry. Therefore, the automotive manufacturers develop Advanced Driver Assistance Systems (ADAS) with the aim of helping drivers in the complex driving task. ADAS are innovative mechatronic vehicle systems that monitor vehicle surroundings, as well as driving behavior [1]. They provide drivers with essential information and take over difficult or repetitive tasks. In critical driving situations, these systems warn and may intervene actively to support the drivers, and hence, lead to increased road safety. ADAS belong to the active safety systems, which help to prevent accidents or at least minimize possible consequences [2].

Using ADAS in cars and trucks has great benefits regarding accident prevention. Reference [2] presented an analysis for thousands of accidents insurance claims in Germany in order to investigate the safety benefits of ADAS. It was found that using one ADAS can prevent up to 45% of a specific type of accident. Therefore, modern vehicles are equipped with various types of sensors, which recognize and analyze the environment. Moreover, the sensory data, which is detected by each sensor can be integrated together to assure its accuracy, and hence, to take appropriate decisions.

Diverse sensor technologies (camera, radar, ultrasonic, etc.) and decision algorithms can provide different levels of assistance [3]. On the one hand, some ADAS, like, e.g., Lane Departure Warning [4], only alert the driver to critical situations by means of optical, acoustic and/or haptic feedback. On the other hand, other ADAS do not only recognize driving situations and warn the driver, but also intervene actively in order to prevent possible collisions. A common example of the latter type is Emergency Brake Assist [5], which applies full braking if driver fails to respond to obstacles in front of the vehicle.

In general, ADAS can be classified according to their functionality in two main categories [3]. Firstly, systems that support the diver and make the driving task easier, like, e.g., navigation devices, night vision systems, and auto-parking systems. Secondly, systems that support the vehicle and make the driving task more safe [3], like, e.g., Adaptive Cruise Control (ACC), Lane Keeping Assistance (LKA), and Lane Change Assistance (LCA).

Automobile manufacturers and suppliers are confronted with considerable technical challenges while developing ADAS. However, there are additional challenging aspects related to ADAS deployment and public acceptance. A flexible test environment is required in order to validate ADAS concepts and assess their decision logic. Clear concepts for driver-vehicle interface have to be addressed in early development phases; this ensures that drivers can handle the systems appropriately. On the other hand, demonstrating safety and comfort benefits of ADAS to consumers is a key factor for smooth market penetration and development.

However, validating and demonstrating ADAS in real traffic environments are impractical and lead to significant efforts and costs. Moreover, real traffic environments are principally random and do not allow for standardized driving tests or reproducible research results. Driving simulators offer a potent virtual prototyping platform to test and verify

ADAS in different development phases [6]. They allow the design, testing, and validation of ADAS in a closed loop together with vehicle components, environment, and driver. ADAS control units and vehicle components could be real, virtual, or a combination of real and virtual components. For demonstration and training purposes, driving simulators can be utilized to make drivers familiar with new ADAS, and hence, accelerate the learning phase.

Driving simulators vary in their cost, structural complexity, and validity from low-level to high-level driving simulators. They extend from driving simulation games for computers or smart phones to highly-sophisticated driving simulators incorporating complex motion platforms and high fidelity visualization systems. ADAS development requires test environments with different levels of details and complexity [7], e.g., Software-in-the-Loop (SiL), Model-in-the-Loop (MiL), Hardware-in-the-Loop (HiL), Driver-in-the-Loop (DiL), etc. For instance, while a SiL environment can be used to test basic ADAS concepts and control algorithms, a DiL environment can be utilized to address the interaction between the driver and vehicle and its systems.

The project TRAFFIS (German acronym for Test and Training Environment for ADAS) is carried out at the University of Paderborn with the target of supporting industrial development, testing and training of modern ADAS using a reconfigurable driving simulator [7]. Despite the fact that the development of driving simulators is costly and complex, available driving simulators in the market nowadays are usually special purpose facilities. They are individually developed by suppliers for a specific task. These driving simulators cannot be reconfigured, or in the best case, they have only some exchangeable components. Only a driving simulator expert can modify the system architecture or exchange one or more components. The existing driving simulators do not allow the system operator to change the system architecture or to exchange simulation models without in-depth know-how of the driving simulator system and its architecture. Therefore, the project TRAFFIS aims to develop a comprehensive environment of reconfigurable driving simulators to support ADAS development. The project is funded by the European Union "ERDF: European Regional Development Fund" and the Ministry of Economy, Energy, Industry, Trade and Craft of North Rhine Westphalia in Germany.

Three driving simulator variants with different complexity levels and simulation fidelity have been built: TRAFFIS-Light, TRAFFIS-Portable, and TRAFFIS-Full. The TRAFFIS-Full variant was first developed for the German Army in 1997 with the aim of performing safety training for the military truck drivers. The Heinz Nixdorf Institute of the University of Paderborn adopted this driving simulator in 2009 in cooperation with Rheinmetall Defence Electronics GmbH. This driving simulator incorporates a complex motion platform, which consists of two dynamical parts with 5 Degrees Of Freedom (DOF) to fully simulate vehicle lateral and longitudinal accelerations. These two parts are independent of each other and the system is fully electrically actuated. The first dynamical part is the moving base. It has 2 DOF and is used to simulate the lateral and longitudinal acceleration of the simulated vehicle. It can move in the lateral plane and at the same time, it has the ability to tilt around the lateral axis with a maximum angle of 13.5 degrees and around the longitudinal axis with a maximum angle of 10 degrees. Four linear actuators are used to control the movements in both directions. The second dynamical part is the shaker system, which has 3 DOF to simulate the roll and pitch angular movements and the heave translation of the simulated vehicle. The shaker is driven by a three drive crank mechanism and by three electrical motors. The driving simulator has an eight-channel cylindrical projection system (powered by 8 LCD-projectors), which covers a 240 degrees horizontal field of view and three displays in order to visualize the simulated rear mirror views. Moreover, the motion platform is equipped with an innovative fixation system, which allows the utilization of different driving cabins, e.g., truck cabin or passenger vehicle cabin, so that drivers experience realistic control cues. The driving simulator is operated by software developed by dSPACE and the University of Paderborn. The software consists of the simulation core, an operator council GUI, a training scenario editing tool, vehicle model, traffic model, and visualization and audio generation components. Figure 1 shows the TRAFFIS-Full driving simulator operated by the University of Paderborn.

Figure 1. TRAFFIS-Full driving simulator operated by the University of Paderborn.

The TRAFFIS-Portable variant has a pneumatic motion platform, which is composed of an actuated inverted hexapod system. A simple motion controller regulates the movements of the motion platform; it is based on virtual vehicle position and orientation. This driving simulator has a four-wall projection system. Figure 2 shows TRAFFIS-Portable driving simulator at the University of Paderborn.

Figure 2. TRAFFIS-Portable driving simulator at the University of Paderborn.

The TRAFFIS-Light variant is simple a PC-based driving simulator with no motion platform. It has a commercial wheel-transmission-pedals set and a racing seat to provide low-cost, but reasonable, physical feedback and control cues. Figure 3 shows TRAFFIS-Light driving simulator at the University of Paderborn.

Figure 3. TRAFFIS-Light driving simulator at the University of Paderborn.

These driving simulator variants, i.e., TRAFFIS-Full, TRAFFIS-Portable, and TRAFFIS-Light, together with an innovative configurability concept offer a flexible test and training environment for various in-vehicle systems [7]. However, the focus is given mainly to the development of ADAS. One particular objective of the project TRAFFIS is the development of a modular simulation environment for ADAS demonstration and training purposes. That is, a simulation framework with flexible prototyping concepts is required for easy and convenient ADAS demonstration and training. As an extension to the work presented in [1], this paper describes the structure of the whole simulation environment utilized in a driving simulator within the project TRAFFIS. Particular focus is given to the module responsible for ADAS simulation and interactive demonstration. Moreover, the main concepts of the visualisation software are presented. That is, the topic of ADAS simulation with driving simulators is addressed thoroughly in this paper.

The architecture of the ADAS virtual prototyping framework is discussed in more details. This framework consists of several functional units enclosing simulation models that were implemented with MATLAB/Simulink. The models are arranged in a modular architecture and developed, so that they communicate in a loosely coupled fashion. The design of the architecture conforms to the configurability concept discussed previously in this section. Adaptation of models interfaces can be performed with minimum effort. The design approach ensures maximum flexibility and scalability for implementing any ADAS virtual prototypes. The design of the functional units is discussed along with input-output relationships of the underlying models. All models are real-time capable, i.e., the simulation runs in real time using the Real-Time Windows Target library from Mathworks.

The developed ADAS simulation framework was integrated with the simulation environment of the TRAFFIS-Light driving simulator, which represents the simplest driving simulator variant within the project TRAFFIS. Furthermore, virtual prototypes of two innovative ADAS are presented to show and validate the capability of the simulation environment and the ADAS prototyping framework for demonstration and training.

This paper is structured as follows: Section II presents related work in the field of ADAS simulation. Section III discusses the modular driving simulation environment, with which the developed ADAS framework was integrated. Section IV presents the design approach of the developed ADAS virtual prototyping framework along with the concepts of its functional units and models. Section V demonstrates two ADAS prototypes realised with the developed framework and demonstrated using the TRAFFIS-Light driving simulator. Section VI derives the conclusion and summarizes the benefits of the presented approach. Finally, Section VII presents the future work with respect to interactive vehicle systems simulations.

II. Related Work

According to literature review, most research work in the field of ADAS simulation considers only the development of specific components, like, e.g., sensor models [8] [9], decision units [10] [11], signal or image processing algorithms [12], etc. On the other hand, there are several commercial solutions for specific ADAS simulation and development. However, a common problem among commercial solutions is the lake of sufficient modularity. In other words, they provide solutions for individual ADAS functionalities. Even if they can be parameterised flexibly to some good extent, adding new ADAS logic and integrating different ADAS functions are typical challenging issues among these commercial solutions.

ASM software from dSPACE provides flexible models for traffic and environment simulations to support the

development and testing of ADAS [13]. Developers can simulate a test vehicle, complex networks, a large number of fellow vehicles, and environmental objects, like, e.g., pedestrians and traffic signs. Moreover, ASM has a graphical user interface to facilitate defining the simulation scenarios and the necessary components.

DYNA4 software from TESIS is a flexible test framework with software and hardware implementations of environment sensors and some defined ADAS functions [14]. It provides overview and automatic comparison of simulation results for further analysis.

CarMaker software from IPG presents an open test platform, which enables a wide spectrum of automotive applications beside ADAS [15]. It offers sophisticated driver model that performs complex driving manoeuvres.

SCANeR Studio software from OKTAL provides a simulation environment to prototype, test and validate some ADAS systems [16]. It includes several sensor models with different levels of complexity. Its high quality real-time visual rendering makes it suitable for camera-based ADAS simulation.

Despite promising work in the research and commercial fields, there is still no comprehensive ADAS simulation platform that can be easily and fast extended to add or integrate new ADAS functions. On contrary, the flexibility and scalability of the developed architecture in this work provide an extensive solution for ADAS simulation and development. Due to its unique modular structure, it presents no limits on the type and complexity of the simulated ADAS functions. The following section describes the developed driving simulation environment; the ADAS simulation framework was integrated with this environment.

III. DRIVING SIMULATION ENVIRONMENT

The simulation environment of the TRAFFIS-Light driving simulator consists of two main functional units: a vehicle dynamics model and a traffic model. Figure 4 illustrates its structure and the direction of information flow.

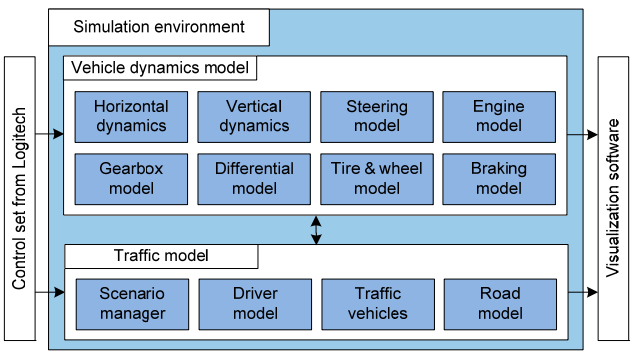


Figure 4. Simulation environment of the PC-based simulator.

Each functional unit consists of real-time capable sub-models implemented with MATLAB/Simulink. The visualization software represents the main feedback cue of the driving simulator. 3D models for the main vehicle, road, and traffic participants are controlled through the corresponding sub-models of the driving simulation environment.

The visualization software was implemented with Unity [17]; a development engine that provides rich and easy functionalities for creating interactive 3D tools. Figure 5 presents sample screen shots for the 3D environment developed with Unity.

Figure 5. Sample screen shots for the 3D environment developed with Unity software.

Night and daylight drives can be performed and the driver can be subjected to different weather conditions, like, e.g., rain, snow, fog, etc. Moreover real test tracks, city streets, and highways can be generated, this is necessary for realistic and engaging driver training. However, modeling real world roads is a cumbersome and time-consuming task. It involves a lot of manual modeling of details along the test track, such as road signs, buildings, vegetation, or other scenery details. Therefore, a method to automate the process of generating models of real roads is utilized [18]. The process uses data from a navigation database to define road sections, from which geometries are generated. This is based on official road construction regulations and guidelines. These geometries are then integrated into models of the surrounding landscape, which are generated from Digital Elevation Models (DEM), aerial images, and Digital Landscape Models (DLM) [18]. Moreover, a procedural rule system for enriching digital terrain with authentic vegetation is used [19]. This procedural approach defines planting rules, which control the placement and distribution of plants

in the scenery based on data from DLM and aerial images [18]. Figure 6 shows a screen shot of a test track with vegetation generation based on the described procedural rule system, it is developed with Unity software [17].

Figure 6. Impression of a test track enriched with vegetation.

Furthermore, realistic sound effects that accompany the 3D models are also used to provide good acoustic feedback cues to the driver. Hence, visual and acoustic information from the ADAS functions are delivered to driver in accordance with traffic situations.

Regarding the hardware and mechanical components, the TRAFFIS-Light driving simulator incorporates a racing wheel-transmission-pedals set from Logitech and a racing seat from Speedmaster. That is, it is still fully interactive with respect to steering, gears, acceleration, and brake controls. This simulator and its simulation environment are considered in this work. The next sub-sections discuss each main functional unit of the simulation environment.

A. *Vehicle dynamics model*

Modeling realistic vehicle dynamics is essential for the development of different in-vehicle systems. In particular, the design of ADAS controllers relies primarily on the underlying vehicle dynamics. The utilized vehicle dynamics model produces the actual physical characteristics of the main vehicle and allows for a total of 16 Degrees Of Freedom (DOF) [20]. The so-called nonlinear double-track model is used for modeling horizontal vehicle dynamics. This model is responsible for 3 DOF: longitudinal and lateral translational motions and a rotational motion around the vertical direction of the road. Figure 7 shows the double-track model, some of the parameters used in the differential equations of this model are also depicted [20].

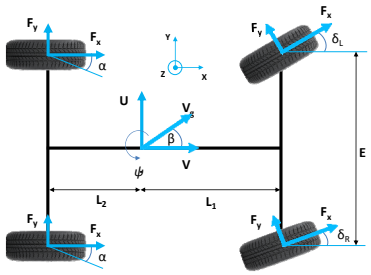


Figure 7. Double-track model for horizontal vehicle dynamics.

In the double-track model, the longitudinal and lateral velocities, as well as the yaw rate of the vehicle are described by a set of differential equations using Newton's law of motion and some basic geometrical relationships [20].

The vertical dynamics of the vehicle depends principally on suspension units at each wheel of the vehicle. The chassis of the vehicle is connected to four wheels through these suspension units. Each unit consists of a simple mass-spring-damper model [21]. Springs and dampers represent the four shock absorbers of the vehicle. The units are constitutively connected through basic mathematical and geometrical relationships [21]. Figure 8 illustrates a sketch for the vertical dynamics of the vehicle.

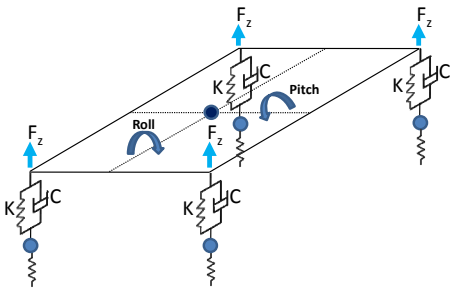


Figure 8. Vertical vehicle model with simple suspension units.

Each wheel has a relative vertical translational motion and a rotational motion around the wheel axis. In addition, each of the front wheels has a relative rotational motion around the vertical direction of the road. The vehicle dynamics model receives control signals from the hardware control set and calculates the resultant motions; these are exported mainly to the visualization software to update vehicle position and orientation on the screen.

The traffic model provides information about the road, i.e., height and friction under each of the vehicle tires; these in turn are used by the vehicle dynamics model to update the calculations of the vehicle position, orientation and speed. The vehicle dynamics model is composed of various sub-models [21]. It implements the blocks shown in Figure 4 as modular Simulink subsystems.

B. *Traffic model*

The traffic model is used to simulate the surrounding vehicles and the road [22]. It simulates realistic behavior of the traffic vehicles and their interactions, which is necessary to give realistic feedback cue to the driver on the one hand, and to efficiently test ADAS functions on the other hand. The traffic model consists mainly of four sub-models: road model, traffic vehicles models, driver model and a scenario manager model. Figure 9 shows these sub-models and their interconnections. The traffic model receives current position, orientation and speed of the main vehicle from the vehicle dynamics model; these are used mainly by the driver

model to arrange for appropriate traffic flow without collisions with the main vehicle.

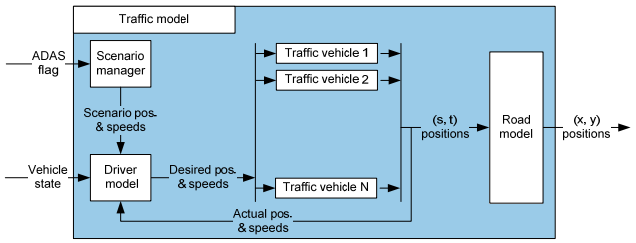


Figure 9. Traffic model and its sub-models.

The road mathematical model is a Matlab function implemented in Simulink, it is responsible for two tasks. The first task is to perform the necessary transformations from local coordinate system (s, t) to global coordinate system (x, y) used by the visualization software. The position of each object within the simulation environment is defined relative to road local coordinate system. However, the visualization software defines each object in 3D world relative to a global coordinate system, which is fixed to the ground. Both are right-hand coordinate systems.

The road consists simply of four straight segments and four round corners. Each road segment has a mathematical description that correlates the (s, t) and (x, y) coordinate systems. Figure 10 shows the geometrical design of the road, the origins of the local and global coordinate systems are depicted together with a numerical example of a sample input (s, t) point and the corresponding output (x, y) point from the road model.

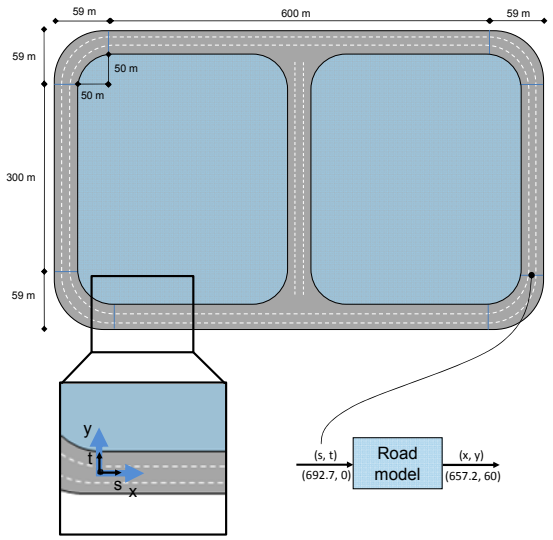


Figure 10. Road geometrical design.

A simple geometrical structure was designed to facilitate the mathematics of coordinate transformation. This also simplifies the implementation of the road 3D model. The second task of the road model is to define the friction 'f' and height 'z' of each point (s, t) of the road. The 'z' position is required by the visualization software for appropriate objects positioning within the 3D world. Both 'f' and 'z' values are required by the main vehicle model; they contribute to the calculations of horizontal and vertical vehicle dynamics, respectively.

Each traffic vehicle model consists of two sub-models: longitudinal direction vehicle sub-model and lateral direction vehicle sub-model. The longitudinal direction sub-model receives the desired s-speed from the driver model, discussed shortly. It calculates the actual s-speed with a smooth transition, which results from a combination of a simple second-order system and a P-controller. The actual s-position of the traffic vehicle is then calculated by integrating the actual s-speed. Similarly, the lateral direction sub-model receives the desired t-position from the driver model. It calculates the actual t-position with a smooth transition, which results from a combination of a simple second-order system and a P-controller. The idea of the traffic vehicle model is to produce smooth and realistic, i.e., not abrupt, movements for the traffic vehicles [22]. This is achieved through the transitional response of the second-order system to unit step inputs of the driver model. The model can be replicated arbitrarily according to the desired number of traffic vehicles. Figure 11 shows a traffic vehicle model and its main connections with the driver model. The calculated (s, t) position of each traffic vehicle is exported mainly to the visualization software to update the position of the corresponding 3D models.

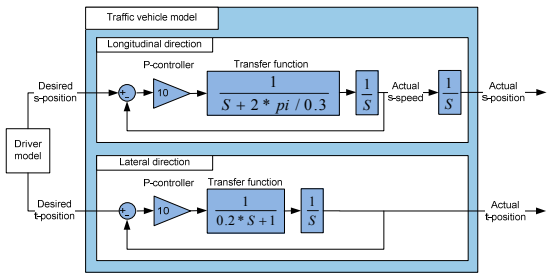


Figure 11. Traffic vehicle model.

The driver model is a Matlab function implemented in Simulink. It arranges for smooth traffic flow by controlling the speeds, and hence the positions, of all traffic vehicles. The driver model calculates and adjusts the speeds according to the current traffic situation. It receives the current (s, t) position of each traffic vehicle as well as the position and orientation of the main vehicle. Accordingly, it monitors the distances between all the vehicles on the road and overrides the default speed values of the traffic vehicles in case of any possible collision. The traffic vehicles have to follow the predetermined longitudinal speed and lateral position given by the driver model.

The scenario manager model is used for moving the traffic vehicles to compose a specific traffic situation, like, e.g., a sudden vehicle incursion from right. It is a Matlab function implemented in Simulink. The scenario manager observes the position and speed of the main vehicle. It moves the traffic vehicles according to a desired predefined

scenario. According to the vehicle systems or functions under test, arbitrarily further traffic scenarios can be added to this model. The driver model receives the vehicle positions and speeds determined by the scenario manager model. According to the current traffic situation, the driver model decides whether to execute the orders of the scenario manager or to override them. The main target is it to achieve the desired traffic scenario with smooth flow and without vehicle collisions. Switching between the different scenarios can be performed during simulation runtime.

IV. ADAS Simulation Framework

The vehicle dynamics model and traffic model constitute the central functional units of a simulation environment for a simple driving simulator. However, a comprehensive simulation framework is still required to conveniently simulate different ADAS functionalities. Active safety in general and ADAS in particular exhibit continuous development. New ADAS functions are developed to achieve safer traffic flow and more comfortable driving. Moreover, the availability of a wide range of sensors and the possibility to integrate different sources of information allow the development of more new reliable ADAS. Hence, one principal requirement for building a flexible ADAS test and training environment is to maintain maximum modularity and scalability. The developed ADAS virtual prototyping framework is structured in a modular form that ensures its scalability. That is, new ADAS prototypes can be added almost without significant input-output interface adjustments. Furthermore, different ADAS can be integrated together to implement more advanced capabilities such as autonomous driving.

Driving is a multitasking activity, where drivers have to manage their attention between various actions and reactions within a dynamic traffic environment [23]. The design approach of the developed ADAS simulation framework is based on an analogy between human driving behavior and the functionality of ADAS. Figure 12 shows the structure of the ADAS simulation framework. It consists of four functional units or stages: user interface stage, recognition stage, guidance stage and control stage. The latter three functional units resemble the activity model the human driver mainly follows while driving a vehicle. The recognition stage represents the senses of human drivers for recognizing road path and other traffic participants, i.e., current traffic situation. The guidance stage corresponds to the reasoning capabilities of the human driver and compromises made according to the recognized traffic situation, i.e., decisions to accelerate, brake, steer or to make a certain maneuver. The control stage simulates the actual physical actions the human driver performs to carry out appropriate decisions.

Related approaches for human driving models are presented in [24] and [25]. The analogical comparison with human drivers is valid under the assumption that any ADAS can be represented as an assisting automatic driver that warns the driver and/or takes over the driving tasks in critical traffic situations.

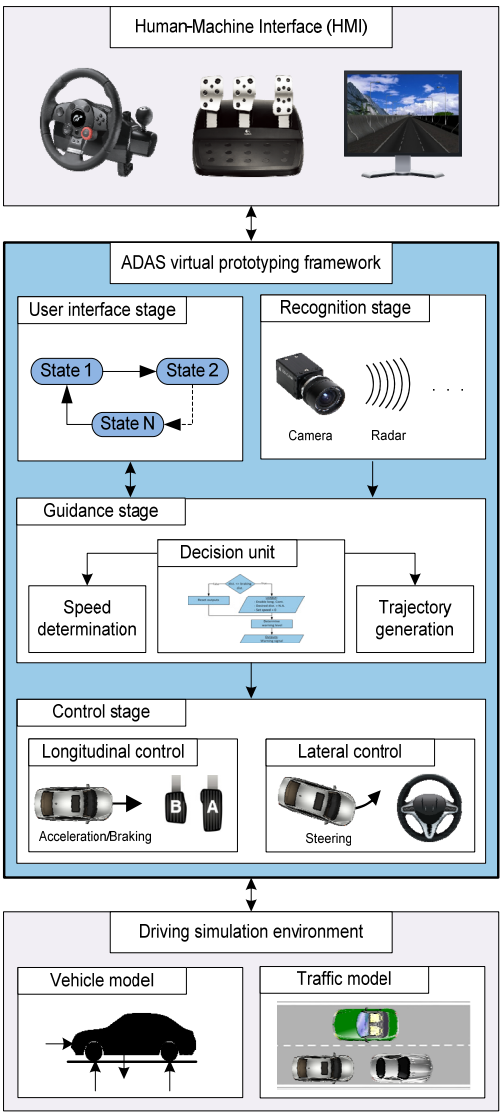


Figure 12. ADAS simulation framework and its relation with the driving simulation environment and HMI.

As shown in Figure 12, the ADAS virtual prototyping framework is connected to the other functional units of the driving simulation environment and the hardware controller set along with the visualization software (HMI) of the TRAFFIS-Light driving simulator. The ADAS simulation framework receives inputs from the HMI to set the ADAS states, i.e., activate, deactivate, or alter some parameters. It eventually applies force feedback on the steering wheel according to the driving situation and the type of the activated ADAS. The ADAS simulation framework gets the states of the main vehicle, i.e., position, orientation and speed, which are calculated by the vehicle dynamics model. In case of ADAS with active intervention, it overrides the requests of the human driver and controls the states of the vehicle. The ADAS simulation framework notifies the

traffic model regarding the activated ADAS, the traffic model invokes in turn predefined traffic scenarios and provides information about the traffic participants. The following sub-sections discuss the design of each functional unit of the ADAS simulation framework and the fundamental input-output signals.

A. User interface stage

The user interface stage accounts for the interaction between user, i.e., simulator driver, and the ADAS simulation framework. It implements the logic required for transitioning between different ADAS functional states, like, e.g., on, off, standby, etc. Each ADAS user interface is modeled separately as a Stateflow sub-model (a control logic tool used to model event-driven systems within Simulink). Figure 13 shows the structure of the user interface stage and the main input-output signals.

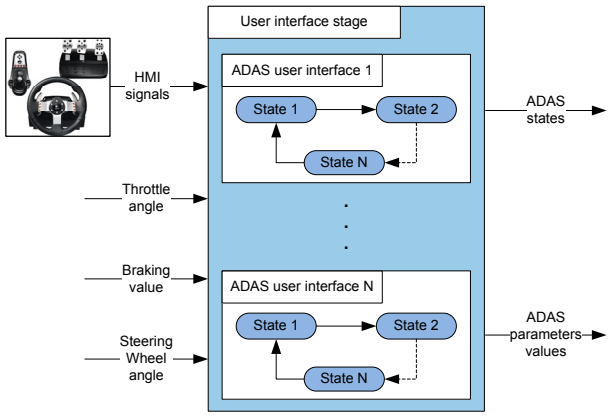


Figure 13. ADAS user interface stage.

Each sub-model receives an enable/disable signal from the buttons set, as well as the values of the acceleration and brake pedals, gear selector and steering wheel of the Logitech controller. Furthermore, it gets feedback signals indicating the desired maneuvers of ADAS controllers, namely, throttle angle, braking value and steering wheel angle. These are compared with corresponding signals indicating the intention of the driver, which is provided through the HMI. If there is a difference, and taking ADAS type into account, the corresponding sub-model decides if ADAS should make a transition from one functional state to another. For instance, while an autonomous driving function will be deactivated if the driver moves the steering wheel slightly; an emergency braking function will not be deactivated for such an action.

As outputs, indications for ADAS functional states along with the desired ADAS parameter values are exported to the corresponding ADAS sub-routines within the guidance stage, discussed in a later section.

This arrangement for the user interface stage conforms to the modularity and scalability requirement of the ADAS simulation framework. For modeling new ADAS, corresponding Stateflow sub-models have to be implemented separately within the user interface stage using the same set of input-output interfaces.

B. Recognition stage

Driver assistance systems require surrounding recognition capabilities to be able to perceive the traffic environment. Any ADAS must incorporate one or more sensors, like, e.g., GPS, cameras, radar, ultrasonic, laser, lidar. Many variants already exist in market; moreover, a lot of new sensor technologies and concepts are being developed, like, e.g., sensor fusion [26]. Hence, there are a lot of sensor models to be integrated in order to achieve a comprehensive ADAS virtual prototyping framework. The recognition stage is composed mainly of two units: a detection unit containing different sensor models and a relevance filter unit. Figure 14 shows the structure of the recognition stage and the essential input-output signals.

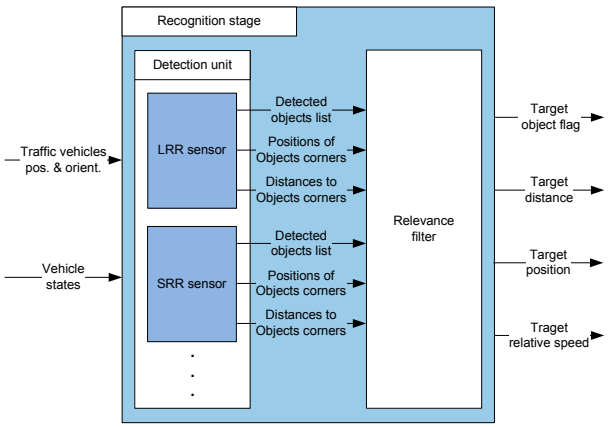


Figure 14. ADAS recognition stage.

Information about road and traffic participants is provided through the traffic model. Vehicle position, orientation and speed, i.e., vehicle states, are provided by the vehicle dynamics model. The detection unit is designed in the form of a bowl that contains different sensor models, like, e.g., radar sensor model, ultrasound sensor model, etc. The main output from a sensor model is a list of objects characterized with detection flags, i.e., detected objects list. In addition, each sensor model provides the positions and distances of detected object corners. Short-Range Radar (SRR) and Long-Range Radar (LRR) sensor models have been implemented within the detection unit. Both models are based on the mathematical description or geometry of detection area [27]. The long-range radar model is ideally suited for detection distance longer than 30 meters; it can typically detect objects 250 meters away. On the other hand, the short-range radar model provides wider view and detection distance below 30 meters. All parameter values can be modified to alter the geometrical description of detection area if necessary, i.e., the geometrical coverage and detection range are adjustable, so that sensor characteristics can be changed arbitrarily.

Within the relevance filter unit, detected objects are further filtered according to the position and orientation of the main vehicle relative to the road. That is, the outputs of all sensor models are forwarded to a relevance filter, which generates a flag indicating the most relevant object to the main vehicle, i.e., target object. Moreover, relative speed of the target object and distance and position of its nearest corner are calculated. Figure 15 illustrates the selection functionality of the relevance filter unit.

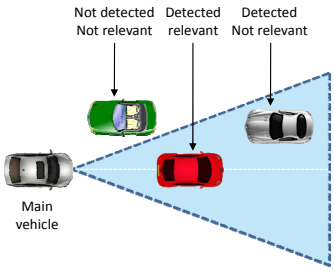


Figure 15. Target object selection of the relevance filter unit.

The detection unit is extensible for additional sensor models to be developed, whereas the functionality of the relevance filter unit has not to be altered. However, the relevance filter unit considers only sensors of the same direction of detection and determines only one target object. If other sensor models for other directions of detection are to be implemented, like, e.g., right and left sides of the vehicle, corresponding relevance filter units have to be designed conforming to the structure of the recognition stage and the same set of input-output signals.

C. *Guidance stage*

As mentioned previously while making analogy between the developed ADAS simulation framework and the human driving model, the guidance stage represents the understanding of recognized traffic situations and the decisions required for safe or comfortable driving. Figure 16 shows the structure of the guidance stage and the main input-output signals.

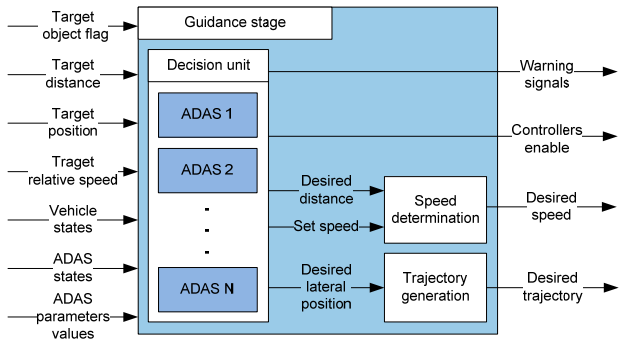


Figure 16. ADAS guidance stage.

The guidance stage derives its central role from being in the middle of a detection phase, i.e., recognition stage, and an action phase, i.e., control stage. On the one hand, it interprets the information provided by the recognition stage, i.e., it evaluates the perceived traffic situations. On the other hand, it determines the actions required to avert undesirable traffic situations.

The guidance stage is consisted of three sub-functions: Decision unit, speed determination and trajectory generation. These sub-functions are discussed next.

- Decision unit

The logic of each ADAS is implemented within the decision unit as a separate sub-routine. The decision unit receives indication for the presence of a target object along with its relative speed, distance to and position of its nearest corner from the recognition stage. Figure 17 shows a flow chart for the main function of the decision unit.

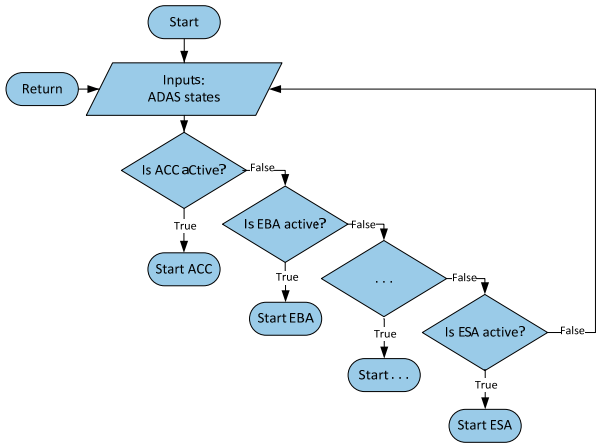


Figure 17. Transition logic between ADAS sub-routines.

The user interface stage implies which ADAS is to be activated with which parameter values. The main function of the decision unit loops through all the implemented ADAS sub-routines. Only that of the chosen ADAS is executed while other ADAS sub-routines are ignored. It considers the traffic situation detected by the recognition stage, ADAS states and parameter values exported by user interface stage and vehicle states provided by vehicle dynamics model. Accordingly, it determines desired distance to a target object, set speed or desired lateral position required to alter the path of the main vehicle. In addition, it sends enable signals to corresponding vehicle controllers, i.e., longitudinal and/or lateral controller, discussed in a later section. The activated ADAS generates warning signals required to trigger some display elements within the visualization software.

Similar to the user interface stage and recognition stage, the decision unit is extensible, so that any logic for new ADAS prototypes can be simply added as new separate sub-routines. The set of input-output signals is comprehensive and suitable for almost all active and passive ADAS.

- Speed determination

This function maintains constant time headway space to a target object that eventually drives with lower speed than that of the main vehicle [28]. Principally, the headway distance varies with main vehicle speed; this allows for a fixed margin in time for the ADAS to react to changes in the speed of the target object. The speed determination function is basically a distance controller that determines the speed required to maintain the desired headway space, taking the speed of the target object into account. It is based on the so-called slide mode control [29]. It is a simple control method that proves good stability especially, where the control actions are discontinuous functions of system states and inputs.

The speed determination function handles the orders of the decision unit with respect to the longitudinal direction. While the desired headway space is provided by the decision unit, i.e., the sub-routine of an activated ADAS, a speed command is generated to obtain this distance accordingly. Figure 18 shows the difference between the desired and actual headway distances.

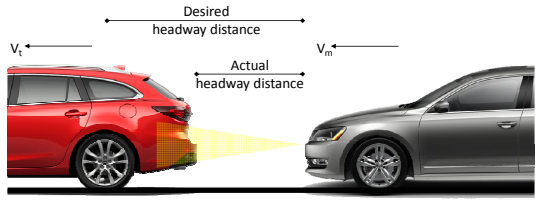


Figure 18. Headway distance control and speed determination.

Moreover, the function selects the minimum of the ADAS set speed, like, e.g., set speed of an adaptive cruise control, and that required for following a target object while preserving constant headway space. Finally, the desired speed is forwarded to the longitudinal controller discussed in a later section.

- Trajectory generation

This function generates the trajectory required to guide the vehicle through the road or to move it from one lateral position to another. The function encloses the mathematical description of the road, so that the generated trajectory reconsiles with road path. The trajectory is generated in the form of a moving point in front of the vehicle. The activated ADAS within the decision unit determines the desired lateral position required to adjust the vehicle path or to avoid a collision for example. The function limits the rate of lateral position change generated within the decision unit in order to obtain reasonable and realistic lateral transitions. Although it handles the orders of the decision unit mainly with respect to the lateral direction, the function adds a predetermined offset to the longitudinal component of current vehicle position. Hence, the location of the moving point is updated continuously and gradually to form the desired trajectory, as shown in Figure 19.

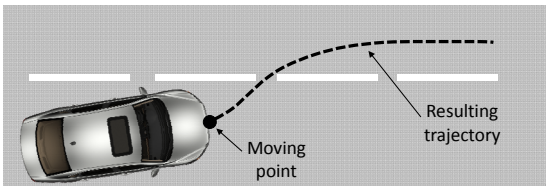


Figure 19. Moving point for trajectory generation.

The desired trajectory represented as postion updates is forwarded then to the lateral controller discussed in a later section.

D. *Control stage*

A motion controller is required in order to control the state of the vehicle in case of active ADAS intervention. As shown in Figure 20, decoupled longitudinal and lateral controllers were implemented to execute the orders of the guidance stage and guide the vehicle accordingly.

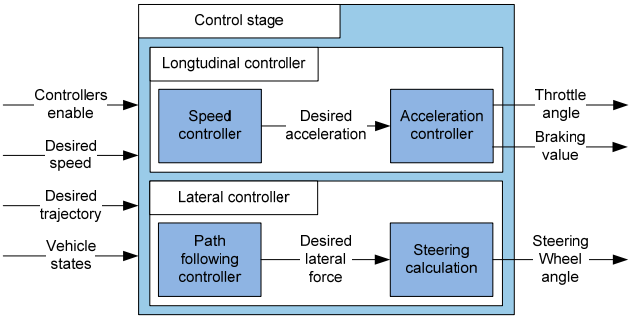


Figure 20. ADAS control stage.

The control stage gets an enable signal from the guidance stage that indicates which controller is to be activated, and hence, moving the vehicle with a desired speed in a desired direction. These controllers are discussed next.

- Longitudinal controller

The longitudinal controller is a cascaded speed-acceleration control loop system [30]. It is composed of two successive controllers: speed controller and acceleration controller. The speed controller is a Proportional-Integral (PI) type that constitutes the outer loop of the longitudinal controller. The speed command from the guidance stage is compared with the actual speed of the vehicle to generate a speed error. The speed controller generates an acceleration value required to overcome the speed error. It is followed by an anti-windup function to prevent output saturation [31]. The desired acceleration is forwarded then to the acceleration controller.

The acceleration controller constitutes the inner loop of the longitudinal controller. The desired acceleration is compared with the actual acceleration of the vehicle to generate an acceleration error. The acceleration controller implements the inverse form of vehicle dynamics and drivetrain of the vehicle model [32]. The acceleration

controller is composed mainly of three sub-models, as shown in Figure 21.

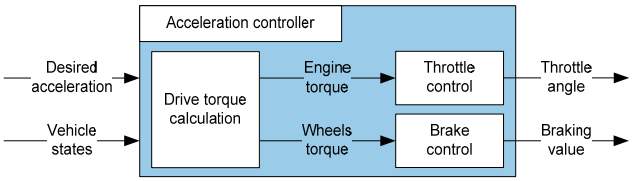


Figure 21. Sub-models of the acceleration controller.

The drive torque calculation sub-model generates the wheels torque and engine torque required to achieve the desired acceleration. It is based on the dynamics equations of the vehicle model. The throttle control sub-model generates the throttle angle according to the required engine torque. It is based on the engine model within the vehicle dynamics model. Similarly, the brake control sub-model generates the braking value according to the required wheels torque [33]. It is based on the braking model within the vehicle model. The longitudinal controller exports the throttle angle or braking value to the vehicle dynamics model. For comfort driving and realistic vehicle behavior, the throttle and brake control sub-models do not allow the acceleration and deceleration to exceed predetermined limits.

- Lateral controller

The lateral controller handles the path following control problem, i.e., how to control the vehicle, so that it can faithfully follow a prescribed path. As shown in Figure 20, it is composed mainly of two sub-models. The path following controller sub-model gets the trajectory generated by the guidance stage in the form of a moving point, i.e., a point directly in front of the vehicle that updates its location on a certain path. It calculates the front axle force required to let the vehicle adjust its orientation, and hence, follow the moving point to pursue the desired trajectory. The path following controller is based on the feedback linearization control method [34]. The basic idea is to convert the closed-loop control system including the plant, i.e., the horizontal vehicle dynamics model in this case, into linear system dynamics. The method was applied to the bicycle vehicle model [20] and showed optimal robustness even at stability borders, such as rapid steering maneuvers or driving at relatively high speeds in sharp curves. According to the horizontal vehicle dynamics, the steering calculation sub-model determines the steering angle, which corresponds to the desired lateral force. Moreover, it calculates the steering wheel angle using the inversion of the steering model within the vehicle dynamics model. Finally, the lateral controller exports the steering wheel angle required to guide the vehicle in the desired direction to the vehicle dynamics model, and hence, following a certain trajectory.

The designed longitudinal and lateral controllers can serve a variety of active ADAS functions, where a spontaneous rapid maneuver or the whole driving task is taken over by an automated intervention. The generality and simplicity of the interface between the developed guidance and control stages make it convenient to develop and plug new ADAS functions. The following section presents the logic of two innovative ADAS functions implemented in the decision unit within the guidance stage.

V. ADAS Prototypical Implementation

The developed ADAS virtual prototyping framework can be used for simulating almost any ADAS function. The recognition stage can be extended for additional sensor models. The guidance stage is also extensible, so that any logic for new ADAS functions can be simply added as new separate sub-routines. The control stage covers the longitudinal and lateral directions, and hence, it can be used principally for any active ADAS.

To prove its usability in general and to show the benefits of its modular structure in particular, prototypes for two new ADAS were implemented: Emergency Brake Assist and Emergency Steer Assist. These systems aim to help drivers to avoid accidents by alerting them to a potential collision and initiating automatic braking or steering maneuver. They represent the state of the art in ADAS development [35]. Although they have different types of intervention, both functions have been implemented without any special interface adjustments due to the modularity and scalability of the ADAS simulation framework described in this paper.

A. Emergency Brake Assist

Emergency Brake Assist (EBA) is an ADAS sub-routine implemented within the decision unit of the guidance stage. The system compensates for failures in the driver's action on the brake pedal. In general, drivers in emergency situations tend to apply insufficient pressure or release braking pressure too early. Figure 22 shows a flow chart for a simplified version of the EBA sub-routine.

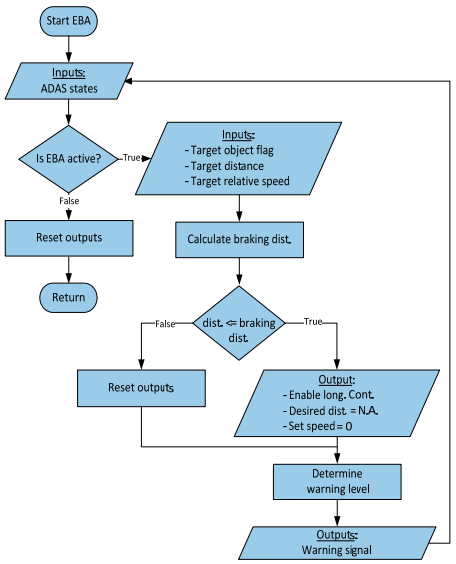


Figure 22. Simplified version of EBA logic within the decision unit.

According to recognized moving or standing objects in front of the vehicle, EBA initiates automatic braking in the case of a potential rear-end collision provided that the driver has not responded to prior warnings signals [36].

The intention of the driver is observed through the user interface stage, and hence, is embedded within the ADAS states signal. The EBA sub-routine gets the distance and relative speed of a target object existing in front of the vehicle from the guidance stage. The critical braking distance, i.e., safe distance, is calculated from the provided inputs. This means, the safe distance is variable and depends mainly on the relative speed of the target object. If the actual distance to the target object gets close to the safe distance within predefined limits, the function initiates optical and acoustic warning signals to be handled by the visualization software.

The optical warning has three levels: a green cautionary signal if the target object ahead is close, a yellow alert signal if the safe distance is reached and a red critical signal if the actual distance is equal to or fell below the safe distance. In the latter case, if the driver fails to take braking or steering actions, i.e., when an emergency situation is fully confirmed and the state of the target object flag does not change, the EBA sub-routine enables the longitudinal controller and sets the speed to zero. The sub-routine overrides the acceleration request of the driver who is effectively taken out of the loop. However, the driver still can retain control anytime by taking an appropriate steering action, and hence; changing the state of the target object flag.

The function was tested and validated with many test scenarios, where different values for the speed of the main vehicle and traffic vehicle ahead were considered. Figure 23 illustrates the switching point between warnings and active intervention distances of the EBA sub-routine

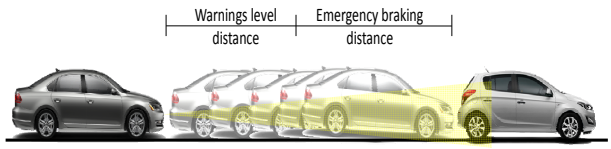


Figure 23. EBA intervention in case of no driver response.

B. *Emergency Steer Assist*

Emergency Steer Assist (ESA) is an ADAS function implemented within the decision unit of the guidance stage. The function supports the driver in the lateral driving task [36]. According to recognized sudden right or left incursion from a traffic object and if the driver has no time left for braking, the function initiates rapid automatic steering intervention in the case of predicted collision, as shown in Figure 24. ESA calculates the optimal trajectory around the appeared object and applies steering torque to help to follow the trajectory and stabilize the vehicle. However, the driver remains in control of the vehicle and can override the system at all times

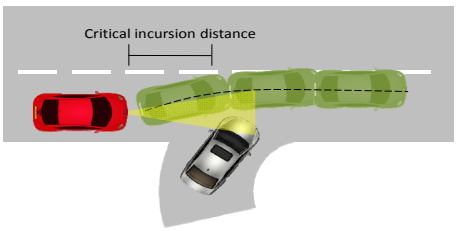


Figure 24. ESA intervention due to sudden road incursion.

Almost similar to the Emergency Brake Assist function, the intention of the driver is observed through the user interface stage. Figure 25 shows a flow chart for a simplified version of the ESA sub-routine.

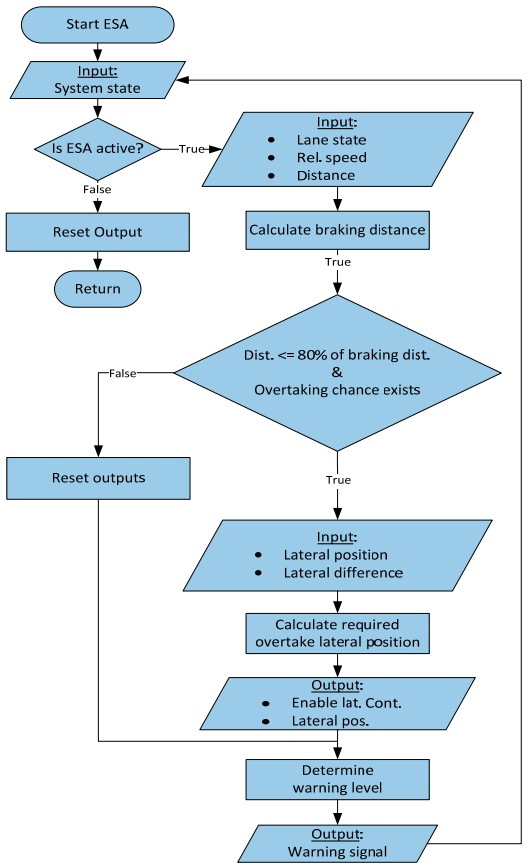


Figure 25. Simplified version of ESA logic within the decision unit.

If a target object appeared suddenly within the lane of the vehicle, the function decides to steer the vehicle abruptly in the opposite direction. This decision takes the form of a desired (x, y) point, which is exported to the lateral controller. The speed of the vehicle, the distance at which the target object appeared and the intention of the driver are factored in the decision of the function. The critical incursion distance is variable and depends mainly on the

speed of the vehicle [36]. Figure 26 shows a screen shot while ESA performs a rapid maneuver to avoid a pedestrian.

Figure 26. ESA performs a rapid maneuver to avoid a pedestrian.

The function was tested and validated with test scenarios, where different values for the speed of the main vehicle, as well as different distances to the incurring target vehicle were considered.

VI. Summary and Conclusion

Advanced Driver Assistance Systems (ADAS) gain importance due to their safety and comfort features. The ADAS virtual prototyping framework described in this paper offers a flexible solution to efficiently validate ADAS concepts and easily demonstrate their benefits to customers. The presented approach is based on an analogy between the functionality of ADAS and the human driving model. This resulted in a comprehensive architecture, which is composed of modular and extensible functional units.

The developed ADAS virtual prototyping framework was integrated with the real-time simulation environment of the TRAFFIS-Light driving simulator. To validate the approach and the capabilities of the developed ADAS simulation framework, prototypical implementation of two innovative ADAS functions was presented. Although both functions show different types of intervention, no special signal interface adjustments were necessary. The design of the other functional units of the simulation environment, i.e., vehicle dynamics model and traffic model, has not to be adjusted for any future ADAS prototypes.

A group of test persons were involved in the behavioral validation process of the driving simulator after integrating the ADAS virtual prototyping framework [37]. In other words, an assessment of how drivers react and perform with respect to the implemented ADAS prototypes has been made. The test persons have been subjected to near collision situations, where different values for the speed of the main vehicle and traffic vehicle ahead were considered. The behavioral validation process showed how the test persons could reasonably handle ADAS warnings and active interventions with very good learning curves. Effectiveness, proper operation and drivers' acceptance of the implemented ADAS were evaluated.

The presented approach added new capabilities to the PC-based driving simulator for assessing ADAS algorithms and performing drivers training by means of a driving simulation environment. In general, the modularity and scalability requirement of an ADAS training environment for the project TRAFFIS was fulfilled.

VII. Future Work

Driving simulators are built to address several aspects in the automotive and transportation fields. They are used for the development of in-vehicle systems, analysis of driving strategies, as well as for demonstration and training purposes.

The majority of available simulators are single-user stand-alone systems. However, as vehicle systems are becoming more complex, driving simulation must keep up in terms of scalability and flexibility. For instance, the significance of C2X-Communication systems has grown in the recent years [38]. These systems allow the vehicles to communicate with other each other, as well as with road infrastructure [39]. Similarly, cooperative advanced driver assistance systems, i.e., interconnecting driver assistance systems of different vehicles on the road, are gaining a lot of attention [40] [41]. These systems benefit from the new communication technologies and the utilization of GPS receivers in vehicles. They add new dimensions of safety, comfort, and optimized traffic flow.

Testing cooperative vehicle systems is even harder than testing traditional stand-alone driver support systems [42]. There are more vehicles and interaction possibilities with each other and road infrastructure. As a potent testing platform, future driving simulation should allow realistic cooperation between different interactive simulation entities, which represent their counterparts in real traffic situations.

Networked driving simulation systems can facilitate this challenge [43]. They allow developers to embed the logic of future cooperative vehicle systems into realistic and interactive traffic scenarios without the effort and costs of real test-drive [44]. Moreover, multi-user driving simulators that communicate with each other can demonstrate realistic effects of driver-driver interaction in more complex simulation scenarios. This is the major motive for extending the developed simulation framework to allow for the simulation of multi-user interactive driving simulation.

The development of such a networked driving simulation system includes many challenging tasks in order to provide a reliable and realistic simulation environment. For example, it requires utilization of the so-called global time management [45] [46]. That is, synchronizing the local time and event processing of each individual driving

simulator in order to guarantee simulation reliability and test accuracy [47].

The next steps aim to develop a concept for a synchronization mechanism for networked driving simulators. This will allow the developed simulation framework to be utilized in a distributed driving simulation system. The synchronization mechanism should facilitate coordination among different participating simulators, which interact within one traffic simulation scenario. In particular, it should guarantee performance and efficiency of the driving simulation system. Finally, the new design approach has to ensure a modular structure that allows easy integration and exchange of driving simulators in a network.

REFERENCES

[1] K. Abdelgawad, M. Abdelkarim, B. Hassan, M. Grafe, and I. Gräßler, "A Scalable Framework for Advanced Driver Assistance Systems Simulation," In Proceedings of SIMUL 2014, the Sixth International Conference on Advances in System Simulation (IARIA), Nice, France, October 2014.

[2] T. Hummel, M. Kühn, J. Bende, and A. Lang, "Advanced Driver Assistance Systems – An investigation of their potential safety benefits based on an analysis of insurance claims in Germany," German Insurance Association - Insurers Accident Research, Research Report FS 03, Berlin, 2011.

[3] J. Golias, G. Yannis, and C. Antoniou, "Classification of driver assistance systems according to their impact on road safety and traffic efficiency," In Transport Reviews - Journal of Intelligent Transportation Systems, Taylor & Francis Group, vol. 22, 2002, pp. 179-196.

[4] P. Hsiao, K. Hung, S. Huang, W. Kao, and C. Hsu, "An embedded lane departure warning system," IEEE 15th International Symposium on Consumer Electronics (ISCE), Singapore, June 2011, pp. 162-165, ISSN: 0747-668X, ISBN: 978-1-61284-843-3.

[5] R. Zheng, K. Nakano, S. Yamabe, M. Aki, and H. Nakamura, "Study on Emergency-Avoidance Braking for the Automatic Platooning of Trucks," IEEE Transactions on Intelligent Transportation Systems, China, August 2014, Vol. 15, No. 4, pp. 1748-1757, DOI: 10.1109/TITS.2014.2307160, ISSN: 1524-9050.

[6] F. Colditz, L. Dragon, R. Faul, D. Meljnikov, and V. Schill, "Use of Driving Simulators within Car Development," In Proceedings of Driving Simulation Conference, North America, Iowa City, USA, September 2007.

[7] B. Hassan and J. Gausemeier, "Concept of a Reconfigurable Driving Simulator for Testing and Training of Advanced Driver Assistance Systems," IEEE Transl. ISAM 2013 China, vol. 2, July 2013, pp. 337-339.

[8] S. Pechberti, D. Gruyer, and V. Vigneron, "Radar simulation in SiVIC platform for transportation issues. Antenna and propagation channel modelling," IEEE Transactions on Intelligent Transportation Systems, Alaska, USA, September 2012, pp. 469 - 474, DOI: 10.1109/ITSC.2012.6338631, ISSN: 2153-0009.

[9] H. Chiang, Y. Chen, B. Wu, and T. Lee, "Embedded Driver-Assistance System Using Multiple Sensors for Safe Overtaking Maneuver," IEEE Systems Journal, November 2012, Vol. 8, Issue 3, pp. 681 - 698, DOI: 10.1109/JSYST.2012.2212636, ISSN: 1932-8184.

[10] L. Yong and L. Xia, "Safety Driving Decision-Making of the AVCSS," IEEE Transactions on Intelligent Computation Technology and Automation, Hunan, China, October 2008, pp. 477 - 481, DOI: 10.1109/ICICTA.2008.171, ISBN: 978-0-7695-3357-5.

[11] M. Horwick and K. Siedersberger, "Strategy and architecture of a safety concept for fully automatic and autonomous driving assistance systems," IEEE Intelligent Vehicles Symposium, California, USA, June 2010, pp. 955 - 960, DOI: 10.1109/IVS.2010.5548115, ISBN: 1931-0587.

[12] L. Genxian, W. Dongsheng, W. Haixia, and L. Zhenyu, "Image Processing Memory Optimization for Multi-camera Based Advanced Driver Assistance Systems," IEEE Transactions on Measuring Technology and Mechatronics Automation, Zhangjiajie, China, January 2014, pp. 313 - 318, DOI: 10.1109/ICMTMA.2014.78, ISBN: 978-1-4799-3434-8.

[13] dSPACE AutomationDesk, Real-Time Kernel (RTK) Real-Time Kernel (RTK) on Models (ASM), DS2211, dSPACE Catalog, dSPACE GmbH, 2009.

[14] D. Block, S. Heeren, S. Kühnel, A. Leschke, and B. Rumpe, "Simulations on Consumer Tests: A Perspective for Driver Assistance Systems," In Proceedings of Engineering Simulations for Cyber-Physical Systems Conference (ES4CPS), Dresden, Germany, March 2014.

[15] S. Ziegler and R. Höpler, "Extending the IPG CarMaker by FMI Compliant Units," In Proceedings of 8th International Modelica Conference, Dresden, Germany, March 2011.

[16] B. Lacroix, P. Mathieu, and A. Kemeny, "A Normative Model for Behavioral Differentiation," In Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Sydney, Australia, 2008, pp. 96-99.

[17] A. Gloria, F. Bellotti, R. Berta, and E. Lavagnino, "Serious Games for Education and Training," International Journal of Serious Games, Vol. 1, No. 1, 2014, pp. 100-105, ISSN: 2384-8766.

[18] L. Rui, D. Burschka, and G. Hirzinger, "Real time landscape modelling and visualization," IEEE International Geoscience and Remote Sensing Symposium, Barcelona, Spain, July 2007, pp. 1820-1823, DOI: 10.1109/IGARSS.2007.4423175, ISBN: 978-1-4244-1211-2.

[19] J. Berssenbrügge, J. Stöcklein, A. Andre and I. Gräßler, "Procedural Generation of Vegetation for a Virtual Test Track," In Proceedings of the International Design Engineering Technical Conferences & Computers and Information in Engineering Conference ASME 2014, Buffalo, NY, USA, August 2014.

[20] H. True, "The dynamics of vehicles on road and on tracks," Swets & Zeitlinger B.V., Lisse, Netherlands, vol. 37, April 2003, pp. 96–105.

[21] R. N. Jazar, "Vehicle dynamics: Theory and application," Springer Science+Business Media, LCC, New York, USA, 2008, pp. 37-279, e-ISBN 978-0-387-74244-1.

[22] J. Barcelo, "Fundamental of traffic simulation," Springer Science+Business Media, LCC, New York, USA, 2008, pp. 15-63, ISSN: 0884-8289, e-ISBN: 978-1-4419-6142-6.

[23] C. Macadam, "Understanding and modeling the human driver," Journal of Vehicle System Dynamics 49, vol. 40, nos. 1-3, 2003, pp. 101-134.

[24] D. T. Mcruer, R. W. Allen, D. H. Weir, and R. H. Klein, "New results in driver steering control models," Journal of Human Factors and Ergonomics Society, 19(4), SAGE Publications, California, USA, August 1977, pp. 381-397, DOI: 10.1177/001872087701900406.

[25] G. A. Bekey, G. O. Burnham, and J. Seo, "Control Theoretic Models of Human Drivers in Car Following," Journal of Human Factors and Ergonomics Society, 19(4), SAGE Publications, California, USA, August 1977, pp. 399-413, DOI: 10.1177/001872087701900406.

[26] R. Altendorfer, S. Wirkert, and S. Heinrichs-Bartscher, "Sensor Fusion as an Enabling Technology for Safety-critical Driver Assistance Systems," SAE International Journal of Passenger Cars - Electronic and Electrical Systems, October

2010, SAE International, USA, 2010, pp. 183-192, ISSN 0148-7191.

[27] T. Akenine-Möller, "Fast 3D triangle-box overlap testing," Journal of Graphics Tools archive, Vol. 6, No. 2, September 2001, pp. 29-33.

[28] N. Benalie, W. Pananurak, S. Thanok, and M. Parnichkun "Improvement of Adaptive Cruise Control System based on Speed Characteristics and Time Headway," IEEE/RSJ International Conference on Intelligent Robots and Systems, Missouri, USA, October 2009, pp. 2403-2408.

[29] J. E. Slotine and W. Li, "Applied nonlinear control," Prentice Hall Englewood Cliffs, New Jersey, USA, ISBN: 0-13-040890-5, 1991, pp. 276–307.

[30] V. V. Sivaji and M. Sailaja, "Adaptive Cruise Control Systems for Vehicle Modeling Using Stop and Go Manoeuvres," International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, vol. 3, Issue 4, July 2013, pp.2453-2456.

[31] C. Poussot-Vassala, O. Senameb, L. Dugardb, and S. M. Savaresic, "Anti-windup Schemes for Proportional Integral and Proportional Resonant Controller," In Proceedings of Power Electronics Conference, Roorkee, India, June 2010.

[32] K. Yi, Y. Cho, S. Lee, J. Lee, and N. Ryoo, "A Throttle/Brake Control Law for Vehicle Intelligent Cruise Control," In Proceedings of Seoul 2000 FISITA World Automotive Congress, Seoul, Korea, June 2000.

[33] C. Poussot-Vassala, O. Senameb, L. Dugardb, and S. M. Savaresic, "Vehicle Dynamic Stability Improvements Through Gain-Scheduled Steering and Braking Control," Journal of Vehicle System Dynamics 49, vol. 00, no. 00, January 2009, pp. 1597-1621, DOI: 10.1080/00423114.2010.527995.

[34] M. Abdelkarim, T. Butz, and A. Moutchiho, "A nonlinear path following controller for lateral vehicle guidance - Ein nichtlinearer Bahnfolgeregler zur Fahrzeugquerführung," Fahrermodellierung in Wissenschaft und Wirtschaft, Fortschritt-Berichte VDI, vol. 22, no. 35, VDI Verlag, Düsseldorf, Germany, June 2013, pp. 135-145, ISBN: 978-3-18-303522-9.

[35] A. Eckert, B. Hartmann, M. Sevenich, and P. Rieth, "Emergency Steer & Brake Assist – A Systematic Approach for System Integration of two Complementary Driver Assistance Systems," Continental AG. Germany: Paper Nr. 11-0111.

[36] C. Keller, T. Dang, H. Fritz , A. Joos, and C. Rabe, "Active Pedestrian Safety by Automatic Braking and Evasive Steering," IEEE Transactions on Intelligent Transportation Systems, December 2011, Vol. 12, Issue. 4, pp. 1292 - 1304, DOI: 10.1109/TITS.2011.2158424, ISSN: 1524-9050.

[37] Z. Mao, X. Yan, H. Zhang, and C. Wu, "Driving Simulator Validation for Drivers' Speed Behavior," In Proceedings of the Second International Conference on Transportation Engineering, Chengdu, China, July 25-27, 2009, pp. 2887-2892, ISBN: 9780784410394.

[38] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2X Communication: Securing the Last Meter - A Cost-Effective Approach for Ensuring Trust in Car2X Applications Using In-Vehicle Symmetric Cryptography," In Proceedings of IEEE Vehicular Technology Conference, San Francisco, USA, Septemper 2011, pp. 1-5, DOI: 10.1109/VETECF.2011.6093081, ISSN: 1090-3038.

[39] S. Boskovich, K. Boriboonsomsin, and M. Barth, "A developmental framework towards dynamic incident rerouting using vehicle-to-vehicle communication and multi-agent systems," In Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems, Funchal, Portugal, September 2010, pp. 789-794, DOI: 10.1109/ITSC.2010.5625251, ISSN: 2153-0009.

[40] Y. Takatori and H. Yashima, "Performance evaluation of vehicle cooperative driving assistance systems that uses forward obstruction detecting sensors and inver-vehicle communication," In Proceedings of IEEE 9th International Conference on Intelligent Transport Systems Telecommunications, Lille, France, October 2009, pp. 622 - 627, DOI: 10.1109/ITST.2009.5399280, ISBN: 978-1-4244-5346-7.

[41] J. Fischer, A. Menon, A. Gorjestani, C. Shankwitz, and M. Donath, "Range sensor evaluation for use in Cooperative Intersection Collision Avoidance Systems," In Proceedings of IEEE Vehicular Networking Conference, Tokyo, Japan, October 2009, pp. 1-8, DOI: 10.1109/VNC.2009.5416389, ISBN: 978-1-4244-5685-7.

[42] Q. Wang and C. Phillips, "Cooperative collision avoidance for multi-vehicle systems using reinforcement learning," In Proceedings of IEEE 18th International Conference on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland, August 2013, pp. 98-102, DOI: 10.1109/MMAR.2013.6669888, ISBN: 978-1-4673-5506-3.

[43] T. Bando and T. Shibata, "Networked driving simulator based on SIGVerse and lane-change analysis according to frequency of driving," In Proceedings of 15th International IEEE Conference on Intelligent Transportation Systems, Alaska, USA, September 2012, pp. 1608-1613, DOI: 10.1109/ITSC.2012.6338804, ISSN: 2153-0009.

[44] Y. Zhao, A. Wagh, K. Hulme, C. Qiao, and W. Sadek, "Integrated Traffic-Driving-Networking Simulator: A Unique R&D Tool for Connected Vehicles," In Proceedings of International Conference on Connected Vehicles and Expo, Beijing, China, December 2012, pp. 203-204, DOI: 10.1109/ICCVE.2012.45, ISBN: 978-1-4673-4705-1.

[45] I. Tacic and M. Fujimoto, "Synchronized data distribution management in distributed simulations," In Proceedings of the 12th IEEE Workshop on Parallel and Distributed Simulation, Banff, Canada, May 1998, pp. 108-115, DOI: 10.1109/PADS.1998.685276, ISBN: 0-8186-8457-7.

[46] P. Galluscio, J. Douglass, A. Malloy, and A. Turner, "A comparison of two methods for advancing time in parallel discrete event simulation," In Proceedings of the IEEE Simulation Conference, Arlington, USA, December 1995, pp. 650-657, DOI: 10.1109/WSC.1995.478840, ISBN: 0-78033018-8.

[47] W. Xuehui , Z. Lei, X. Nong, and T. Yuhua, "Time Management in Parallel Discrete Event Simulation," In Proceedings of the IEEE International Forum on Information Technology and Applications, Chengdu, China, May 2009, pp. 209-212, DOI: 10.1109/IFITA.2009.96, ISBN: 978-0-7695-3600-2.

HiPAS: High Performance Adaptive Schema Migration

Development and Evaluation of Self-Adaptive Software for Database Migrations

Hendrik Müller, Andreas Prusch, and Steffan Agel
Pasolfora GmbH
An der Leiten 37, 91177 Thalmässing, Germany
{hendrik.mueller|andreas.prusch|steffan.agel}@pasolfora.com

Abstract – **HiPAS stands for "High Performance Adaptive Schema Migration" and is a self-adaptive software system, aimed at reducing downtime during offline database migrations by automatically adapting to available system resources. The process of a database migration can be shortened by parallelizing the data transfer up to a certain degree. In this article, we describe how HiPAS was enabled to continuously adapt the parallelization degree according to its operational environment in order to avoid both overloading and idle resources. To automate the developed method, we implemented HiPAS following decisions taken within the dimensions of design space for self-adaptive software. Based on a centralized control pattern in distributed systems, HiPAS uses a feedback loop to enable adaptions. Hence, according to monitored system information, the current utilization is adjusted whenever necessity is assumed. To enable a flexible adaption, the total amount of migration data is partitioned into equal sized transfer jobs, which are distributed across available instances and networks. HiPAS is invoked on database layer and controlled by a temporarily created autonomous database user. Therefore, migration metadata are stored inside tables and highly integrated with the actual migration data. HiPAS was designed and evaluated iteratively following the IS research framework and reveals significant downtime reduction potential compared to non-adaptive migration approaches like Oracle "Data Pump". Our results serve as a contribution for all practitioners, who seek to perform database migrations within a challenging timeframe, as well as researchers on self-adaptive software and their various fields of application.**

Keywords-Adaptability; Anticipation; Self-Adaptive Software; Database Migration; Parallelization.

I. INTRODUCTION

The rapid technical developments inside changing markets, as well as the need for efficiency enhancements, mainly driven by cost pressure, require an occasional transfer of running information systems into a new environment, which fulfills the operational requirements in a more suitable way. This process is referred to as software migration [1], [2] and meanwhile the software's availability can be limited depending on the chosen migration method. Regarding this, basically two approaches can be differentiated:

- online Migration: continuous availability
- offline Migration: interrupted availability

In some critical environments, a downtime is not acceptable, thus online migrations need to be performed. This article deals with the variety of cases, which do not require a costly and complex online migration and a planned downtime is tenable. In that case, the main concern is to keep the downtime as short as possible since the duration of unavailability may result in opportunity costs. In particular, we target migrations applying the "big-bang" strategy [3], thus data is fully migrated at once in contrast to incremental migrations. Since the legacy system (source system) is shut down during the data transfer, starting the target system, referred to as cut-over [4], cannot be performed before all required data has been transferred to the target system's database. The length of downtime depends on the migration approach taken. For database migrations, different system layers can be involved determining the performance and granularity of data selection (see Section II). We investigated the applicability of adaptive capabilities for database migration software in order to reduce the necessary downtime by parallelizing data transfer up to an optimal parallelization degree, which will be continuously adapted to the system's load capacity. Prior tests indicated that overloading the target or source systems resources leads to a temporary stagnation of the whole migration progress, whereas a low utilization wastes available resources, thus underachieving existing downtime reduction potential. In this manner, we contribute to the field of self-adaptive software and support the statement by de Lemos et al. that "self-adaption has become an important research topic in many diverse application areas" and "software systems must become more versatile, flexible, dependable […] and self-optimizing by adapting to changes that may occur in their operational contexts, environments and system requirements" [5].

Moreover, the developed approach "HiPAS" (High Performance Adaptive Schema Migration [1]) is intended to provide dependability and interruptibility, since migration software should be able to identify where to resume an interrupted migration process instead of starting from scratch avoiding the necessity of rescheduling a planned downtime.

Further technically conditioned features will be added in Section III as consequences of the preliminary considerations. Section IV summarizes HiPAS´ architecture by means of introducing adaptability challenges of the subsequent described migration process (Section V). The adaptive capabilities are outlined in Sections VI and VII.

Finally, in Section VIII, we evaluate HiPAS, which refers to both the designed migration method and the migration software, currently implemented in Oracle PL/SQL syntax comprising 8,540 lines of source code.

II. Present Migration Approaches

As introduced previously, migration approaches can be differentiated regarding the availability of the migrated systems into online and offline migrations. For stated reasons, we focus on offline migrations, which can be further classified concerning their own characteristics and their applicability for certain database characteristics:

- invocation layer
- support for change of platform
- support for change of endianness
- support for change of character set
- downtime proportionality

To obtain an overview of present migration approaches, we classified existing available methods using the example of Oracle databases and the above enumerated characteristics:

TABLE I. Classification of Present Migration Approaches.

Migration Method	Invocation Layer/ Granularity	Downtime Proportionality	Platform Change	Endianness Change	Character Set Change
Storage Replication	Storage/ Storage	negligible	no	no	no
Transportable Database	OS/ Database	Database Size	yes	no	no
Transportable Tablespaces	OS/ Tablespace	Tablespace Size	yes	no	no
Cross Platform Transportable Tablespace	OS/ Tablespace	Tablespace Size	yes	yes	no
Transportable Tablespaces using Cross Platform Incremental Backups	OS/ Tablespace	Data Alteration Rate	yes	no	no
Oracle-to-Oracle (O2O)	OS/ Schema	Amount of Migration Data	yes	yes	no
Datapump	Database/ Value	Amount of Migration Data	yes	yes	yes
Export/Import	Database/ Value	Amount of Migration Data	yes	yes	yes

As shown in Table I, the divergence of the source and target database in terms of platform, endianness and character set technically limits the available migration methods.

A critical decision criterion for the remaining contemplable methods is the demand for downtime shortness resulting in lower opportunity costs during the unavailability of the database and all relying applications. The fact that a high throughput for data transfer was achieved as yet by eliminating upper layers and protocols, leads to the conflicting goals of flexibility and performance when selecting a migration method. The lower a layer a migration is invoked on, the more flexibility is lost, since changes of database characteristics might not be supported and the possible granularity for migration data selection decreases. Finally, downtime proportionality refers to the entity, which the downtime length depends on; this can be the amount of migration data or the data alteration rate if incremental methods are used.

When designing HiPAS, we pursued the goal of achieving a short downtime and at the same time providing the flexibility of migrating between divergent databases and selecting the data as granular as possible. This was achieved by invoking the migration on database layer without ever leaving this layer during the whole migration process and by parallelizing the data transfer adaptively in respect of the system's resources. Therefore, we add "adaptability" as a further decision criterion for migration software capabilities.

III. Preliminary Considerations

The performance of migration software highly depends on how well its design fits to the operating environment and the intended range of functions. Previous system and data analyses are necessary to conclude with a migration design, which has been aligned to the findings in multiple iterations following the guidelines of design science in information system research [6]. Figure 1 shows how the designed artefact HiPAS is related to its environment and knowledgebase base inside the information systems research framework.

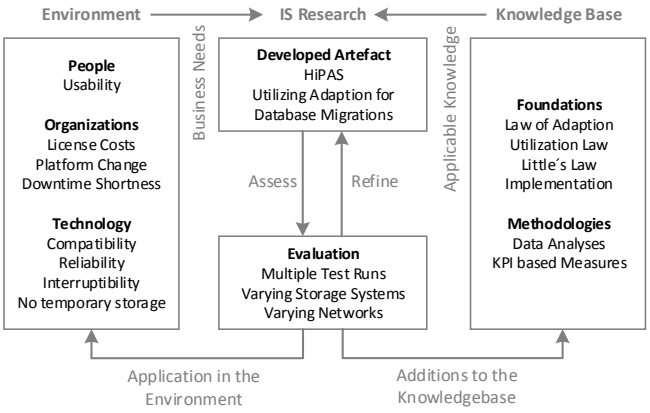


Figure 1. HiPAS as an IS Research Artefact (Adapted from Information Systems Research Framework [6]).

HiPAS was intended to be built upon findings of preliminary analyses (Knowledgebase) described in this section, as well as business requirements (Environment) and from then on has been improved continuously, based on evaluation runs performed in a variety of different environments provided generously by customers.

A. Enterprise Data Structures

When moving existing data files to the target system, as migration approaches invoked on storage and database layer do (see Section II), the valuable downtime is partly spent migrating unnecessary or useless data. The allocated size of a data file implies unused space and indexes. To gain an overview of typical storage occupancies, we analyzed 41 SAP systems productively running at a German public authority by querying the allocated disk space, the used disk space and the space used for indexes with a result shown in Figure 2.

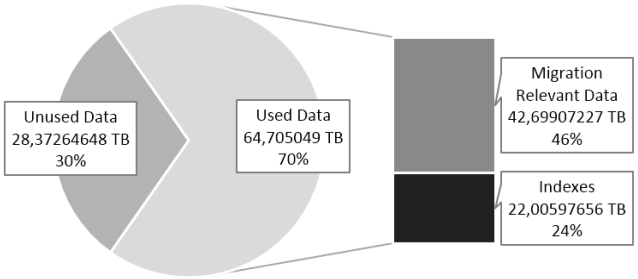


Figure 2. Average Structure of Allocated Data.

For these 41 SAP Systems, we identified an overall amount of 93.08 TB allocated data. From this amount, about 28 TB (30 %) represented allocated space, which was not yet filled with data. From the used space of 65 TB, about 22 TB (24% of the overall amount) were filled with indexes. The remaining 43 TB (46% of the overall amount) represent the actual relevant data, which necessarily needs to be transferred into the target database within a migration. Indexes can be created at the target system and do not have to be transferred, thus saving network bandwidth. Depending on the layer the migration is invoked on, unused but allocated data can be excluded as well.

In this case, if all of the analyzed SAP systems needed to be migrated, migration tools not supporting data selection would utilize all involved system resources for transferring data, of which approximately 54% is useless on the target system. Invoking a migration method on software layer, enables both excluding useless data autonomously and implementing self-adaptability.

B. Endianness

When performing a database migration, the byte order in which the source and target system store bytes into memory needs to be considered. This byte order is referred to as endianness and data is stored into data files accordingly, so the endianness can affect the amount of available migration methods and the overall needed downtime.

A major part of migration demanding customers served by the authors of this paper currently initiate migration projects due to licensing and maintenance costs, this amount is strongly influenced by an increasing number of platform migrations from Solaris to Linux, requiring subsequent migrations on upper layers such as the databases tier. The latest International Data Corporation (IDC) report on worldwide server market revenues substantiates this observation by stating that Linux server revenue raised from 17% in Q4 2010 to 23.2% in Q2 2013 compared to Unix decreasing from 25.6% down to 15.1% [7]. The Unix-based Solaris operates on processors following Oracle´s SPARC architecture, whereas Linux distributions can be used on systems based on Intel processors. When migrating from Solaris to Linux, the endianness changes accordingly from big endian to little endian, so the data files cannot simply be moved without converting them before or after the transfer as shown in Figure 3.

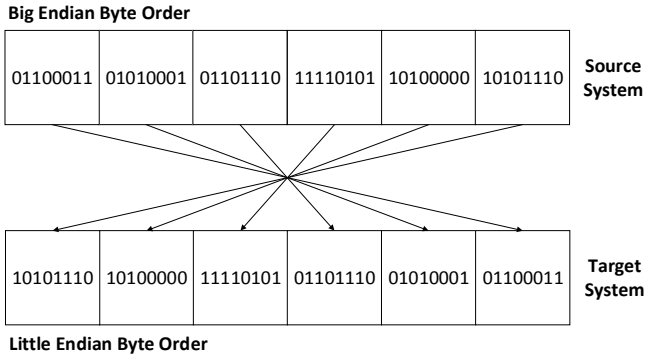


Figure 3. Change of Endianness between Source and Target System.

Alternatively to converting data files, the database migration can be invoked on a layer, which supports saving the data into new files on the target system, such as export-import-tools and HiPAS do. In this case, migration performance can be enhanced by means of adaptive capabilities.

C. Storage I/O Controller

As a consequence of the requirement for downtimes as short as possible, a utilization degree of the underlying storage systems has to be achieved, which enables short response times. The overall amount of requests inside a system (N) equals the product of arrival rate (a) and average response time (R) as expressed by Little´s Law [8]:

$$N = a \times R \tag{1}$$

In addition the Utilization Law [9] defines the utilization (U) of the I/O controller as the product of arrival rate and average processing time (R_S):

$$U = a \times R_S \tag{2}$$

By combining these relations, it becomes clear that the response time depends on the I/O controller's utilization as described within the following formula [10]:

$$R = \frac{R_S}{1 - U} \tag{3}$$

The relation shows that the response time does not change linearly to the utilization. At higher utilizations, the response time grows exponentially as clarified in Figure 4.

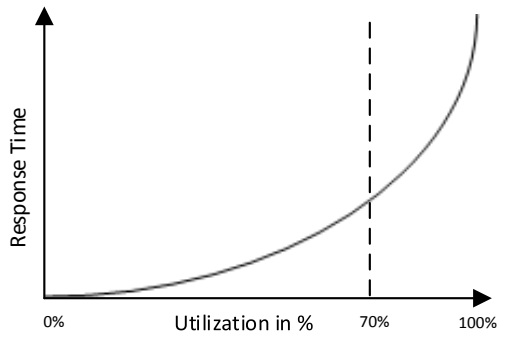


Figure 4. Relation of Utilization and Response Time.

By adapting to the source and target system resources, HiPAS continuously changes the utilization of the I/O controller in order to achieve an optimal relation of response time and utilization supporting the shortest possible overall duration. The storage manufacturer EMC generally describes an average utilization of 70% as optimal [10].

IV. HiPAS Architecture

Following the goals introduced in Section I, we designed the HiPAS migration method as describes in the following.

A. Everything is a Tuple

When performing an automated and controllable migration, a number of interim results arise, e.g., during the analysis of source data. Keeping these information, as well as logging and status information is necessary for the administrator to manage and verify the migration and for the software itself to handle parallel job executions autonomously. The necessity for saving and querying migration metadata leads to HiPAS's design paradigm of not leaving the database layer during the whole migration process. Interim results such as generated DDL and DML Statements for later execution are represented by tuples of tables inside a temporary migration schema enjoying advantages of the databases transactional control mechanisms. The paradigm of everything being a tuple is emphasized by the following list:

- objects to create are tuples (table "cr_sql")
- data to transfer are tuples (table "transfer_job_list")
- running jobs are tuples (table "mig_control")
- parameters are tuples (table "param")
- logs are tuples (table "logging")

After a migration has been performed, its success and the transferred data´s integrity have to be verified. Since logging information was stored during the whole process inside the logging table, SQL can be leveraged to query for certain transferred objects or states or both. For optimizing the migration process, sorting, calculating and analytical capabilities of SQL are utilized, thus, there is no need for any other migration application on operating system level than the database management system (DBMS) itself.

B. Adaptability and Dependability Problems

When designing the migration method and implementing the related software, several challenges had to be faced. In this section, we will briefly introduce some of the most interesting problems and their intended solutions:

- Utilization Problem
- Knapsack Problem
- Distribution Problem
- Dependency Problem
- Index Problem

Subsequently described solution approaches for the above listed problems will provide an overview of the conceived migration method. In-depth sections are referenced.

1) Utilization Problem: Utilization cannot be planned generally since systems behave differently depending on their resources and further running processes. We varified this statement during the evaluation phase by performing migration test runs that have a preliminary defined static parallelization degree. This leads to the risk of both overloading a system and on the other hand not utilizing idle resources. Derived from the relationship between utilization and response time described in Section III-C, Figure 5 shows how the overall performance, in terms of transfer time, behaves at increasing parallelization degrees:

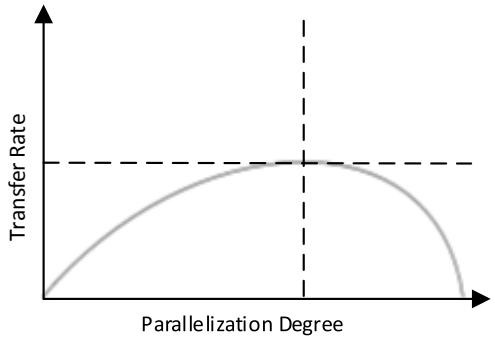


Figure 5. Expansion when Overloading the Storage System.

By choosing the currently optimal parallelization degree adaptively at any time, HiPAS targets an optimal and dynamic utilization, which leads to the shortest possible transfer phase. In this way, we reduced the risk of utilizing the systems too much or not enough. Parallelization is implemented by means of background jobs started through the database scheduler. In this way, the yet manual task of finding the optimal parallelization degree for the respective system environment is intended to be done by HiPAS automatically and adaptively, implicating the ability to change this value dynamically during the whole transfer process.

2) Knapsack Problem: From an amount of objects, defined by their weights and values, a subset with limited weight and maximum total value has to be chosen [11]. This knapsack problem reflects the challenge of choosing optimal combinations of different sized tables to transfer in parallel, since the available computing resources are limited. Large tables should be preferred in a way of starting their transfer at the beginning of the migration process, because a possible failure can require a restart of the table transfer thus delaying the whole migration when started too late. HiPAS circumvents the knapsack problem by dividing large tables into equal sized partitions, which can be transferred in parallel. This offers flexibility in scheduling the data transfer and dynamically adapting the current parallelization degree.

3) Distribution Problem: Depending on the migration environment, the accruing work load can be distributed on multiple instances of a cluster. In terms of network bandwidth, multiple database links can be created on different physical network connections between the source and target system. In this case, HiPAS will distribute data to be transferred equally on the available database links in order to utilize the total available bandwidths. In case of a real application cluster (RAC), HiPAS distributes running transfer jobs on the available instances. Then the fact of the previously mentioned partitioning of large tables needs to be considered. We optimized the data buffers of the instances by distributing transfer jobs, which continue a large table, to the instance, which already transferred previous parts of the same table to avoid reloading the table into multiple buffers of different instances. The corresponding algorithm is explained in Section VII-D.

4) Dependency Problem: When invoking the migration on database layer, dependencies among the transferred objects need to be considered for the transfer order. Surely, users need to exist before importing data into created tables and granting permissions found in the source schema. Constraints like foreign keys have to be disabled temporary, so HiPAS does not have to spend time for calculate a strict and inflexible transfer order. If reference partitioning was used inside the source schema, a parent table needs to exist before the child table can be created following the same partitions. For considering such dependencies, HiPAS calculates a transfer schedule in the first place. Since possible existing triggers will be transferred as well, they need to be disabled during the migration process in order to avoid unexpected operations on the target system, e.g., invoked by an insert trigger.

5) Index Problem: Indexes can either be created directly after table creation or after the table has been filled with data. When creating the index before data load, they will be built "on the fly" during the transfer phase, in contrast, after data load, an additional index buildup phase would need to be scheduled. The right time for indexing depends on the target storage system and network bandwidths. In case of a highly powerful storage system, it might be reasonable to build the indexes directly during data import since the network represents the bottleneck of the whole migration and the storage system would idle otherwise. On the other hand, storage systems can be overloaded when indexes have to be created at import time. Consequently, the decision about the indexing time is another use case for the adaptive capabilities of HiPAS explained in Section VII-C.

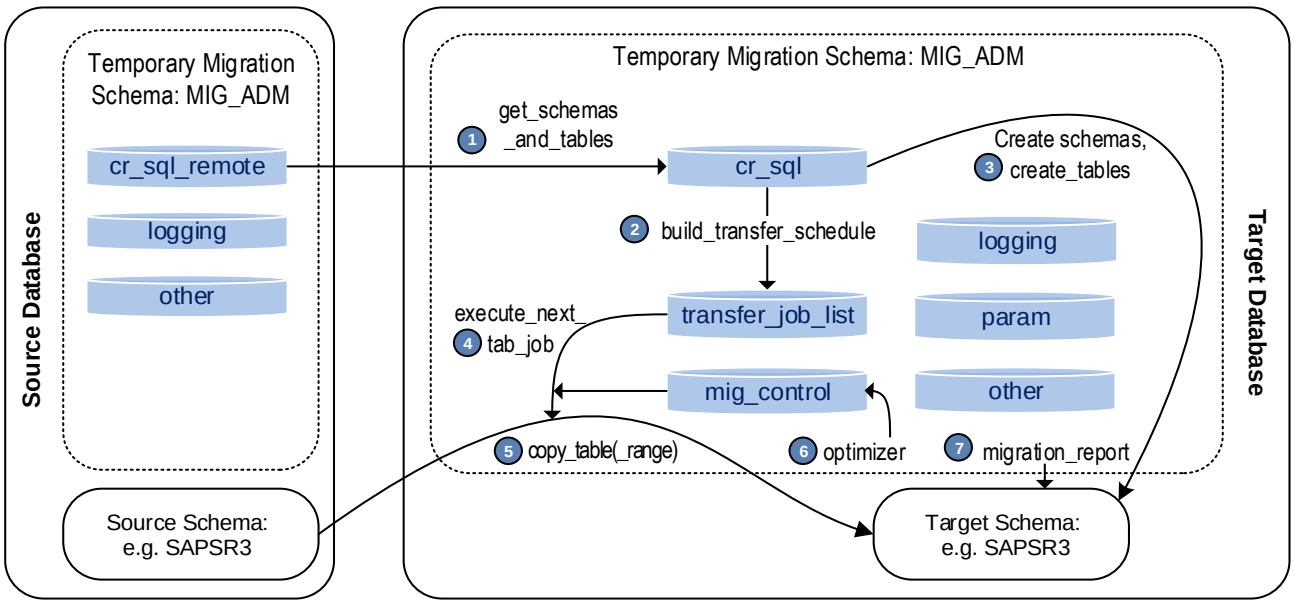


Figure 6. HiPAS Architecture.

V. COMPONENTS AND MIGRATION PROCESS

Assuming that both source and target database system have been physically connected preliminary and are configured to be accessible by each other, the migration process consists of three main phases invoked on the target system, which are briefly described subsequently:

1. Installation and Pre-Transfer (Step 1-3)
2. Adaptive Data Transfer (Step 4-6)
3. Post-Transfer and Uninstallation (Step 7)

Figure 6 shows the steps of these phases, which are invoked on the target system.

A. Installation and Pre-Transfer

Following the paradigm of not leaving the database layer, an additional and temporary schema is created inside both source and target database during an automated installation phase. All subsequent operations will be done by the owner of this schema. Creating this user, as well as creating and compiling a PL/SQL package, needed for performing the migration, is part of an automated installation process. Prior to the data transfer phase, the source schemas need to be analyzed and accordingly created inside the target database. For this purpose, SQL statements for creating the identified objects will be generated and stored inside the table "cr_sql_remote". This table will be copied to the target site and contains information regarding the objects to be created and its creation status. In addition, every operation performed causes status information to be written into the table "logging" (see Figure 7), enabling the database administrator to perform any necessary analysis, e.g., by querying for possible errors during or after the migration:

```
select logdate, loginfo from logging where info_level
= 'ERROR';
```

After the initial analyses of the source schema, all identified objects have the status "init" and, therefore, will be created by HiPAS at the target site. All objects containing "created" inside their corresponding status column will be ignored, enabling the whole migration process to be paused and continued at any time. The table "param" (see Figure 7) serves as a user interface for parameterizing HiPAS manually beforehand, in case certain adaptive capabilities shall not be utilized.

Techniques like reference partitioning inside the source schema have to be considered and will determine the order of creation, since child tables will not be created and partitioned unless the related parent table exists. The Index creation is either part of the pre-transfer or will be initiated after all tables are filled with data. HiPAS decides automatically for the most suitable approach depending on the storage system and network bandwidth as described in Section VII-C.

B. Adaptive Data Transfer

The data transfer is based on two simple SQL statements:

- Insert into a table as selecting from a source table
- Querying remote tables through a database link

The combination of these statements makes it possible to fill local tables with remotely selected data. The resulting command is generated and parameterized at runtime:

```
sql_stmt := 'insert /*+ APPEND */ into "' || schema ||
'"."' || table_name || '" select * from "' || schema
|| '"."' || table_name || '"@' || db_link;
```

This statement is generated and executed by transfer jobs. The number of transfer jobs running in background is adapted continuously and depends on the resource utilization. As a pre-transfer stage, metadata of all objects stored in the source schema has been inserted into a table named "transfer_job_list". Tables to be transferred, exceeding a defined size, will be partitioned and, thus, transferred by multiple transfer jobs. In this case, the job type changes from "table" to "table_range" and row IDs mark the range's start and end (see Figure 7).

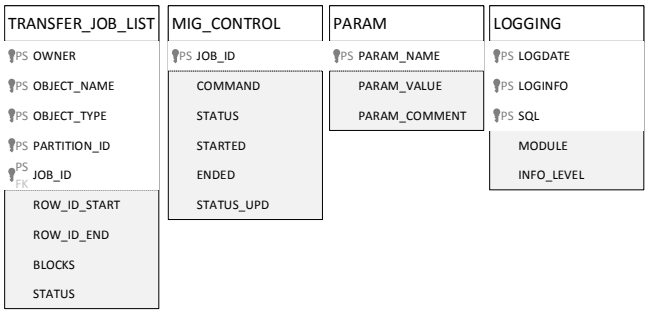


Figure 7. Metadata Entities for the Adaptive Data Transfer Phase.

Through partitioning, HiPAS can adapt more flexible to the current utilization, since the number of parallel jobs can be reduced or increased more frequently. HiPAS' table "mig_control" (see Figure 7) lists all background jobs transferring the objects stored in "transfer_job_list". In this respect, the column "command" inside "mig_control" serves as an interface for controlling the transfer process, either autonomously by HiPAS or manually by the database administrator. When overwriting its content with keywords like "stop" or "continue", individual jobs will be stopped after finishing or continued, causing timestamps to be written into the column "status_upd" and if necessary into "ended". By this means, HiPAS is able to reduce or increase the number of parallel running transfer jobs transparently in respect of the optimizer's decision, which is described in Section VII. For the migration time, all constraints will be disabled temporary by HiPAS, enabling the table "transfer_job_list" to be ordered by blocks instead of considering key dependencies. Existing database triggers will also be disabled avoiding any unintended execution during the database migration.

C. Post-Transfer

After all source data has been transferred into the target schemas, the data has to be validated. Documenting data consistency and integrity is mission critical both for target

database operation and for legal reasons. Only after verifying the equality of source and target data, the migration can be declared as successful, requiring HiPAS to not only compare source and target sizes, but also counting the rows of all tables. Finally, the disabled constraints and triggers will be enabled again.

VI. ENABLING PARALLELIZATION

In order to control the degree of HiPAS utilizing the available hardware resources the migration data transferred at the same time must be limitable. Restricting the number of parallel processed tables would be inappropriate since it required similar sized tables. Instead, a defined number of blocks form a pack of data and a certain number of packs can be processed at the same time. That is, each pack has the same size and will be transferred by a single transfer job. Thus, adding or removing a transfer job burdens respectively disburdens the source and target system. HiPAS adapts to the underlying system resources by deciding autonomously how many transfer jobs are possible at any time.

To enable the amount of data to be partitioned into equal packs, a so called block split range defines their size. Since the tables on the target system are filled by generated "insert as select"-statements, its scope can be limited to a range between two row IDs, which represent the beginning and the end of each data pack. During the source schema analysis, these row IDs are identified by an analytical function. In this manner, large tables are partitioned into groups with row ID boundaries as Figure 8 shows exemplary.

```
GRP MIN_RID            MAX_RID
--- ------------------ ------------------
  2 AAAig0AAAAAABGAAAA AAAig0AAAAAABh/CcQ
  0 AAAig0AAAAAAACAAAA AAAig0AAAAAAAl/CcQ
  1 AAAig0AAAAAAAmAAAA AAAig0AAAAAABF/CcQ
```

Figure 8. Assigning Row IDs as Group Boundaries.

The identified IDs will be used during the transfer phase to limit the data of a single transfer job to the given block split range by adding a "where rowid between"-clause when selecting from the remote database:

```
insert into schema.table_name select * from
schema.table_name@db_link where rowid
between MIN_RID and MAX_RID;
```

Having partitioned the full amount of migration data into parts of a maximum defined size (block split range), HiPAS creates equally treatable transfer entities. These entities can be parallelized up to a degree defined by an adaptive transfer optimizer.

VII. ADAPTIVE CAPABILITIES

For parallelizing the data transfer during phase 2 of the migration process (see Section V-B) with an optimal parallelization degree, we target an adaptive migration software. Adaptivity in general describes the capability of adjusting to an environment. In biology, the term is often used to describe physiological and behavioral changes of organisms in process of evolution. In informatics, the term is transferred to systems or components, which adapt to their available resources. However, here not to increase reproduction chances but often in order to achieve an optimal system performance. Adaption improves the resource efficiency and flexibility of software-intensive systems and means that a system adapts to changes of its environment, its requirements and its resources [12]. According to Martín et al. adaption can also be seen as the first of three stages of the currently conceivable system complexity extent. Anticipation and rationality follow as further stages [13] (see Figure 9).

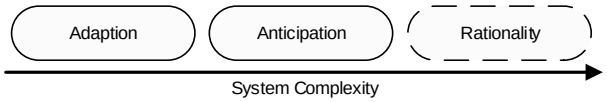


Figure 9. Levels of System Complexity (Adapted from [13]).

Thus, adaption describes the interaction of two elements: A control system and its environment. The goal is to reach a defined state of the environment by means of actions initiated by the control system [14]. The control system then reacts on the self-precipitated changes of the environment with initiating new changes. It has been defined that an adaptive system is present, if the probability of a change of a system S triggered by an event E is higher than the probability of the system to change independently from the event:

$$P(S \rightarrow S'|E) > P(S \rightarrow S') \text{ [13]} \quad (4)$$

Furthermore, the condition has to apply that the system reaches the desired state after a non-defined duration. This implies the convergence of the mentioned probabilities towards infinite:

$$\lim_{t \rightarrow \infty} P_t(S \rightarrow S'|E) = P_t(S \rightarrow S') \text{ [13]} \quad (5)$$

This law of adaption [13] requires the control system to know for each modification of its environment a sufficiently granulated attribute, which contributes to the desired state´s achievement allowing the adaption to end. For the first stage of complexity, the direction and the extent of modifications are built upon each other, thus, enabling the system to reach the desired state incrementally. If the modifications are not steps of a targeted adjustment process, but based on knowledge, predictions [15] or intuition, the process can be defined, in terms of system complexity, as anticipation. The third stage "rationality" implies intelligence; those systems are able to react to unpredictable changes of their environment and to balance contradictory objectives against each other [13]. This stage exceeds the objective of this paper and, therefore, was not scoped. Applying this differentiation on the design of an adaptive migration software, two approaches emerge for parallelizing the data transfer:

- A solely **adaptive** system, based on an incremental adjustment process, until changes do not evoke further improvements, thus, reaching the state of an optimal parallelization degree.
- An **anticipatory** system, which makes continuously new modification decisions independently of each other, based on knowledge about used and monitored resources.

These two approaches have been designed and implemented as described subsequently and evaluated as described in Section VIII. Due to HiPAS' scalable architecture, the respective procedures could be implemented as plugins and additionally started for evaluation. Both plugins control the data transfer via values inside the table "mig_control" (see Section V-B) serving as an interface.

A. Adaption

The solely adaptive approach will successively increase the parallelization degree and, therefore, the source and target systems utilization. After each enhancement its consequences on the system environment meaning the migration performance is measured in terms of inserted megabytes per time unit. The adaption can be started by running an additional procedure "calibrate", which invokes either the procedure "increase" or "decrease" for modifying the parallelization degree, starting from one transfer job per database link at the same time. The number of jobs to be added or deducted will be reduced after each time a change in direction was required, by this means the algorithm brings the number of parallel jobs closer to the optimum. After reaching a defined modification count (number of jobs to add or deduct) the algorithm assumes having approximated the optimal parallelization degree and the adaption ends, representing the finiteness requirement of adaptive systems.

The variable "diff_level" describes the current modification extent, meaning the number of jobs to start additionally or to stop after finishing. To reach a required level of flexibility for changing the number of jobs shortly, the size of a transfer job is limited to the introduced block_split_range. The following code example shows how the number of jobs is reduced by the value of the variable "diff_level":

```
update mig_control set command = 'STOP' where job =
'loop_while_jobs_todo' and command = 'continue' and
rownum <= diff_level; commit;
```

Since the tuples inside "mig_control" represent background jobs and each tuple has a row number, jobs can be stopped for each row number being smaller than "diff_level". The mentioned value "loop_while_jobs_todo" is the name of the procedure every background job runs for processing all defined transfer jobs listed inside the table "transfer_job_list". If a background job is marked with the command "STOP", it will be deleted after finishing the current transfer job and afterwards marked with the keyword "ended".

B. Anticipation

If the adaption is based on predictions, we call it anticipation as the next level of complexity [13]. In contrast to the solely adaptive approach, HiPAS now optimizes the parallelization degree during the whole data transfer phase and based on a different algorithm. For mapping the described theoretical insights to our migration use case, we implemented an optimizer package, which predicts the optimal amount of parallel running jobs for the upcoming period. In this manner, we developed self-adaptive software, for which several definitions exist.

The anticipation-based version of HiPAS complies with a widely referenced definition [16], which was provided by the Defense Advanced Research Projects Agency (DARPA) in an Agency Announcement of 1997: "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible" [17]. In particular, we address the identification of possible performance improvements. Another definition is given by Oreizy et al.: "Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation" [18]. The operating environment in our case is formed by the relevant components of the source and target database system identified in the "Observation" Paragraph.

Properties of self-adaptive software were introduced by the IBM autonomic computing initiative in 2001 [19]–[21]; these are known as eight self-properties. According to Salehie and Tahvildari, this classification serves as the de facto standard in this domain [16]. On major level the following self-properties exist:

- Self-configuring
- Self-healing
- Self-optimizing
- Self-protecting

Salehie and Tahvildari provided a list of research projects in the domain of self-adaptive software from academic and industrial sectors, and classified these projects according to their supported self-properties. It was stated that the majority of the analyzed projects focus on one or two of the known self-properties [16]. When developing HiPAS, we strongly focused on self-optimizing capabilities, which are demanded by customers of our domain. In addition, some self-healing properties for transfer job interruptions are supported. In the following paragraphs, we explain the chosen design decisions for HiPAS according to the dimensions of design space for adaptive software [22] and the optimizer's functionality by running through the phases of one adaption loop, also referred to as feedback loop or MAPE-K loop (see Section B-2).

1) Design Space

Design decisions about how the software will observe its environment and perform adaptions were defined by Brun et al. as design space for self-adaptive systems: "A design space is a set of decisions about an artifact, together with the choices for these decisions. […] A designer seeking to solve a problem may be guided by the design space, using it to

systematically identify required decisions, their alternatives, and their interactions" [22]. The following dimensions of design space were outlined by Brun et al.:

- Observation
- Representation
- Control
- Identification
- Enabling Adaption

Following this systematic approach, we describe the main decisions within these dimensions.

a) Oberservation

Observation is concerned with information about the external environment and the system itself, which needs to be observed by the system [5]. Therefore, Hinchey and Sterrit distinguish between environment-awareness and self-awareness [23], whereby environment-awareness is also called context-awareness [16], [24]. Based on the primarily intended property of self-optimization with respect to performance, we identified the following environmental components, which need to be observed:

- Storage system of the source database system
- Storage system of the target database system
- CPU resources of the source system's instances
- CPU resources of the target system's instances
- Memory resources of the source system's instances
- Memory resources of the target system's instances

The instances of both source and target database system use shared storage, but dedicated computing resources like CPU and memory, therefore, the monitored information must be analyzed on both global and instance-level.

To support system-awareness, the software needs to be aware of the overall migration progress, the number of currently running transfer jobs on each instance, and the status of any job at any time. For implementing self-healing capabilities, information about failed jobs needs to be stored along with respective log information in order to identify root causes and enable retry decisions.

Another important decision is related to the time of observation, which highly depends on domain knowledge about the expected frequency of environmental changes. According to our experiences with job modifications that need to be triggered manually, we implemented a timer, which triggers a cyclic observation every two minutes.

b) Representation

The identified information, which needs to be observed, is represented by performance values and monitored by the database management system. The DBMS stores these values inside performance views, which can be queried by HiPAS. The following values represent the necessary information to base adaption decisions on it:

- Concurrency events on target system
- Concurrency events on source system
- Average write time on target system
- Average read time on source system
- Average read time on target system
- Average write time on source system
- Redo log buffer size
- Available memory size

In order to be self-aware, HiPAS stores all status information regarding running and pending jobs in tables as described in Section V. These tables can be queried by adaption plugins such as the HiPAS optimizer.

c) Control

Control related decisions determine the involved control loops and their interaction, as well as the computation of enacted changes for each adaption [16]. According to [22], different patterns for interacting control loops exist. HiPAS is based on a hierarchical relationship between one master that is located on the target system and multiple slaves represented by instances of the DBMS. This architecture results in a variation of the Master/Slave pattern [22], where the master is responsible for global monitoring, analyzing and planning, and the slaves for instance-specific monitoring and executing (see Figure 10).

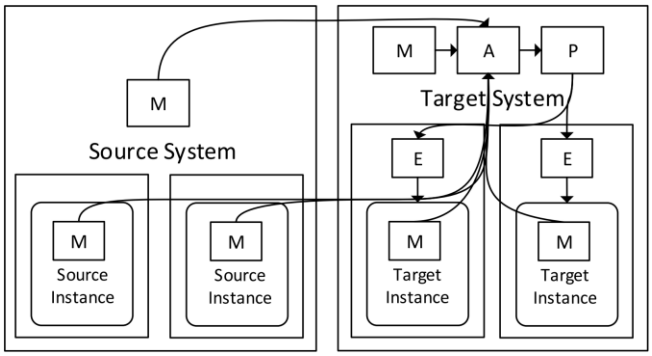


Figure 10. Variation of Master/Slave Control Pattern (adapted from [22])

The master aggregates and analyzes all collected information and then calculates a plan, which includes instance-specific commands. This plan is executed only against the target instances, since data is pulled by transfer jobs running on the target system. Reducing the amount of jobs will reduce the utilization on both target and source system. Hence, we decided for centralized control in a distributed system, which is comparably easy to realize, since all information that is required for performing adaptions is available at the target system through database links [22].

d) Identification and Enabling Adaption

The identification dimension deals with identifying instantiations of the self-adaptive system, which describes the system structure and behavior at a specific point in time [5]. HiPAS' system state is defined by the set of values for the observed performance attributes and the number of transfer jobs running on each target instance. Information about running jobs is stored inside a control table named "MIG_CONTROL" (see Figure 7). This table serves as a central interface for enabling adaptions because it allows plugins such as the HiPAS optimizer to access and change job information at runtime as described earlier in Section V. In the next paragraph, we explain how adaptions are performed by running through one adaption loop.

2) Adaption Loop

The relevant system performance attributes, which represent the information identified for observation, is continuously monitored by the DBMS across all involved database instances.

The optimizer analyzes the enumerated values and calculates a fail indicator, as well as the number of additionally possible jobs according to the measured available resources like memory size and disk utilization. In contrary, the fail indicator indicates possible bottlenecks and can prompt the optimizer to reduce the amount of currently running jobs. The introduced components form a feedback loop according to the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) loop reference model developed by IBM [21] as shown in Figure 11.

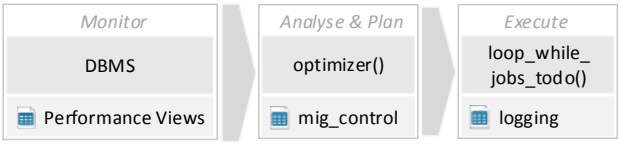


Figure 11. MAPE-K Based Adaptive Feedback Loop.

Typical indicators for possibly arising bottlenecks are increasing concurrency events while the redo log buffer size decreases. An important concurrency event, for instance, occurs when the high water mark of a segment needs to be increased, since new blocks are inserted into the same table by multiple and competing processes, this is known as high water mark enqueue contention [25].

If such a situation has been monitored, the optimizer will reduce the number of parallel jobs based on a high failure indicator. Whenever the optimizer acts, a log string is written to the logging table as in the following example:

```
"Prev Jobs: 40/ Jobs: 40 Max Jobs: 400 # Read Avg: 3.32(20-40) # Write Avg: 105.9(100-200) # R_Read Avg: .12(20-40) # R_Write Avg: .3(20-40) # R Fail Ind: 3 conc:3026(2607) redo:5720763732(5776886904) r_conc:5157(5069) # numjobs > 0 # Jobs being stopped: 0 # (Resource Overload) and numjobs > minjobs and jobs_being_stopped = 0 # Running: 20/Stopping: 5 on inst:1 # Running: 20/Stopping: 5 on inst:2"
```

In the above extracted example, 40 jobs are running in parallel. Due to increasing concurrency events, the optimizer detects a possible overload of the target system and decides to stop 5 running jobs on each instance. The jobs will terminate after they completed transferring their current objects. This is implemented by writing "stop" commands into the table "mig_control" (see Figure 7), which the procedure "loop_while_jobs_to_do()" will carry out. The next log string will start with the information "Prev Jobs: 40/ Jobs: 30" accordingly. Additionally, not only the overall amount of jobs is measured, but also the memory each server process allocates. This value highly depends on the data types of the currently transferred data. If too much memory is allocated, the number of jobs will be reduced as well. In order to avoid downward or upward spirals, e.g., due to the reducing redo log buffer size when stopping jobs, bottom lines and limits are defined. Hence, the optimizer decides on the basis of a branched search for indicating relations between the monitored information. Surely, these are only indicators not to be seen as evidence, so the algorithm follows a heuristic approach. In contrary to the solely adaptive approach and to a statically parallelized transfer, the optimizer is able to dynamically react to unexpected events and predict a possibly optimum level of system utilization during the whole migration process. In the following sections and for the evaluation, when mentioning the adaptive capabilities, we always refer to the anticipatory approach as it was performing more efficient during preliminary tests.

C. Time of Indexing

As previously termed as the "index problem", the right time for indexing the data depends on the combination of storage system performance and network bandwidth. If not manually parameterized inside the "param" table, HiPAS decides by means of test tables filled with random data and having indexes on multiple columns, if it creates the indexes before or after data loading. For the two possibilities of index creation, the time for performing the respective steps is measured and compared to each other. After comparing the two measurements, HiPAS updates the parameter "index_while_transfer" inside the "param" table autonomously by inserting "true" or "false". This test can be performed during a common migration test run on the actual system environment and excluded for the productive migration reusing the "param" table.

D. Transfer Order and Instance Affinity

The table "transfer_job_list" contains all objects, which need to be transferred to the target. When selecting the next object for transfer, this table needs to be ordered by blocks since large objects are preferred by HiPAS. Furthermore, an instance prefers table partitions of tables, which already have been started to be transferred by this instance. Accordingly, the next table or table range to be transferred is always selected as follow:

```
select * from transfer_job_list where status = 'PENDING' and object_type = 'TABLE' and (instance = 0 OR instance = sys_context('USERENV', 'INSTANCE')) order by instance desc, blocks desc, partition_name;
```

If an instance starts transferring a table range of a large table, it marks all other table ranges of the same table by inserting the instance number into all tuples related to this table. By this means, instances reserve tables in order to avoid loading the same table into data buffers of other instances. For this reason, instances prefer tuples marked by themselves and tuples not reserved by any other instance, which has been implemented by means of the above displayed "where clause". In addition, the actual block split range, defining the limit for the size of all table ranges, is identified partly adaptively. For a given maximum block split range, HiPAS calculates the optimal block split range by counting tables and their sizes resulting in an optimal ratio of a ranges size and its total count.

VIII. EVALUATION

The migration method has been tested in several customer environments with differently powerful server, storage systems and networks. Following the design science approach, HiPAS has been improved in multiple iterations based on test results.

A. Experiment Setup

For this paper, we set up a test environment consisting of a source and target system installed on physically separated virtual machines, each having 4 CPUs and 16 GB of main memory. Both the source and target database are real application cluster (RAC) environments running Oracle Database 11g Enterprise Edition Release 11.2.0.3.0. On each side two instances are available connected to the other side through a 1 Gigabit Ethernet. The source system reads from solid state drives and the target system writes on common SATA disks. For evaluation, we performed multiple test runs belonging to the following three different main tests:

(1) Function test with a 300 GB schema (Test A)
(2) Performance test with a 16 GB schema (Test B.1)
(3) Performance test with a 32 GB schema (Test B.2)

To create the different database schemata, we implemented a software package, which generates database schemata filled with random data and including all special cases we could imagine HiPAS to encounter at productive customer environments. By means of this software, we created different sized test schemata inside the source database for test migrations. For the function test (Test A), the schema included characteristics like foreign key constraints, a variety of character, numeric and binary data types, reference partitioning, indexes, table clusters, views, as well as different rights and roles. In this manner, we were able to test the compatibility of HiPAS with different data types, objects and complex data structures. The used schema has an overall size of 300 GB, which was large enough to analyze HiPAS adaptive behavior during the migration run. To compare HiPAS migration performance with the current Oracle standard migration tool for exports and imports "Data Pump" [26], [27], we reduced the size for being able to perform multiple test runs and to average out performance values across all performed runs. These performance focused migration runs are referred to as test B. After each migration, we fully deleted the migrated schema and rebooted the whole server in order to have the same initial cache situation for all runs. The results of all tests are shown subsequently.

B. Results

In the following the results of the function test (A) and the performance tests (B) are presented.

1) Function Test (Test A): As described in Section VI-A, "Test A" aims at analyzing HiPAS adaptive behavior and compatibility. We implemented a package, which compares the created target schema with the original source schema by counting rows and columns. We verified that all data objects were created inside the target schema successfully. The optimizer, providing the adaptive capabilities of HiPAS, writes log information whenever an adaption is needed. An example of such a single log string has been introduced in Section VII-B. Analyzing all tuples, written into the logging table during a migration run, leads to the migration process shown in Figure 12. The transfer started at 12:08 pm and ended at 12:47 pm. HiPAS transferred the created test schema, filled with 300 GB of random data, starting with 20 background jobs running in parallel meaning 10 jobs per instance, since HiPAS identified two available instances on the target system for job distribution. After 39 minutes, the transfer ended with a current total number of 116 parallel running jobs. The "block split range" was 25120 blocks, so, with a configured data block size of 8 KB, each job transferred a maximum amount of approximately 200 MB.

Tables, smaller than the split range, were not partitioned and transferred at the end of the migration, since large tables are preferred by the data selection algorithm. If a job transfers less data (small table), more parallel jobs are possible, so HiPAS raised the number of running jobs as the migration time goes by, which explains the slope of the graph shown in Figure 12.

In a different test using the same schema, we monitored the network interfaces in order to evaluate how the 1 Gigabit Ethernet is utilized by HiPAS. Figure 13 shows the number of kilobytes received and transmitted by one of the physical interfaces, which was monitored using the Linux command "sar".

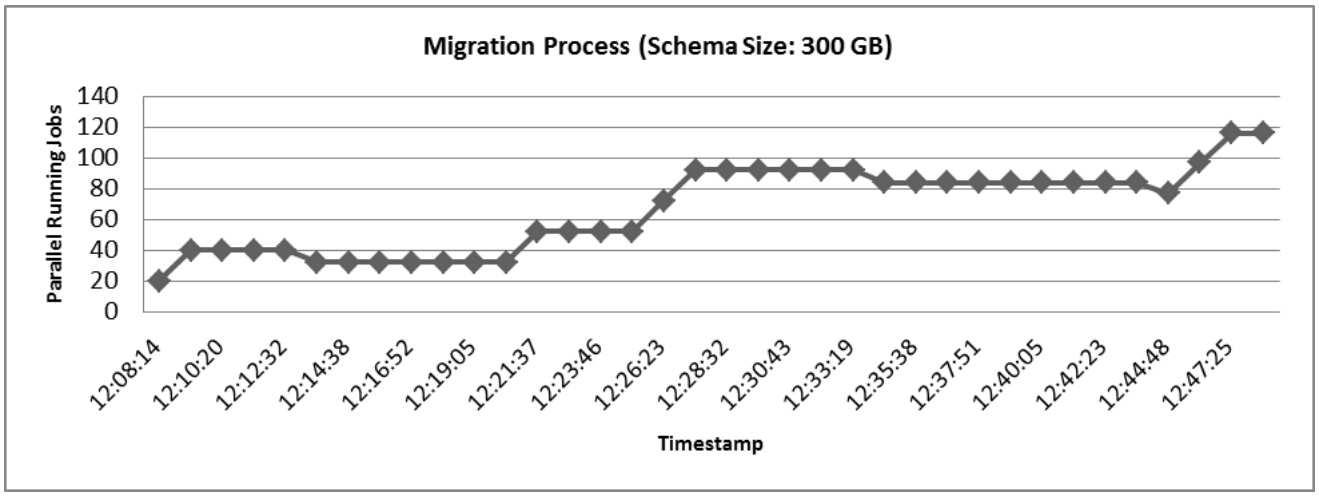


Figure 12. Adaptive Migration Process with HiPAS

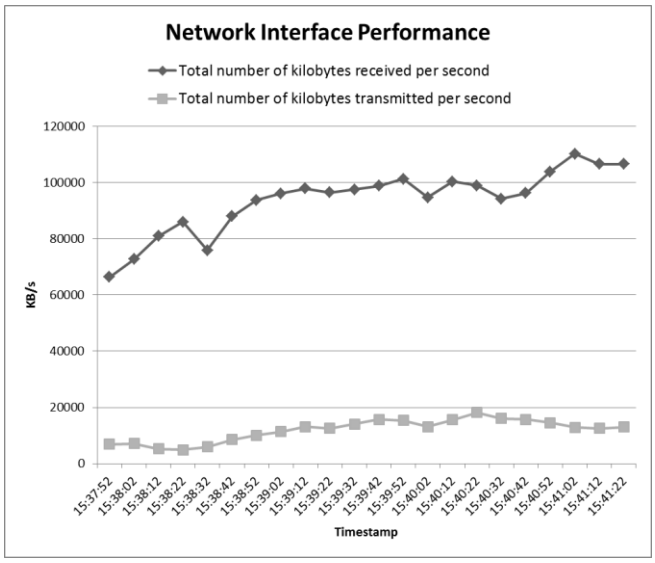


Figure 13. Network interface Performance (Excerpt)

During the monitored timeframe, the network interface, at its highest utilization, received up to 110138.34 KB per second. At that time 12849.31 KB were transmitted, resulting in a total amount of 122987.65 KB/s.

2) Performance Test (Test B.1): For the first performance test, we created a schema of 16 GB including the mentioned data types in Section VIII-A. The different test runs of test B.1 are described as follows:

(1) Migration by means of HiPAS adaptively and with enabled partitioning of large tables
(2) Migration by means of HiPAS with a static parallelization degree of 20 running jobs and enabled partitioning of large tables
(3) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and enabled partitioning of large tables
(4) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and disabled partitioning of large tables
(5) Migration by means of HiPAS without parallelization (sequential) and with disabled partitioning of large tables
(6) Migration by means of Oracle Data Pump

We performed the described test runs three times in order to compensate statistical outliers, possibly caused by uninfluenceable events of the database management system or the operating system. This was necessary because the test runs had to be performed successively to provide the same environment for all tested methods. Afterwards, we calculated for each method the average total duration of the three runs. The final result is shown in Figure 14. The small test schema of 16 GB has been transferred by HiPAS averagely within 11 minutes, enabling adaptive capabilities (more precisely "anticipation") and partitioning of large tables. Transferring the same schema by means of the Oracle tool Data Pump, using the number of available CPUs as the "parallel" parameter [26], took averagely 53 minutes, which means a deceleration of approximately 382% compared to HiPAS. Comparing the different HiPAS migration runs with each other, it can be stated that parallelizing in general

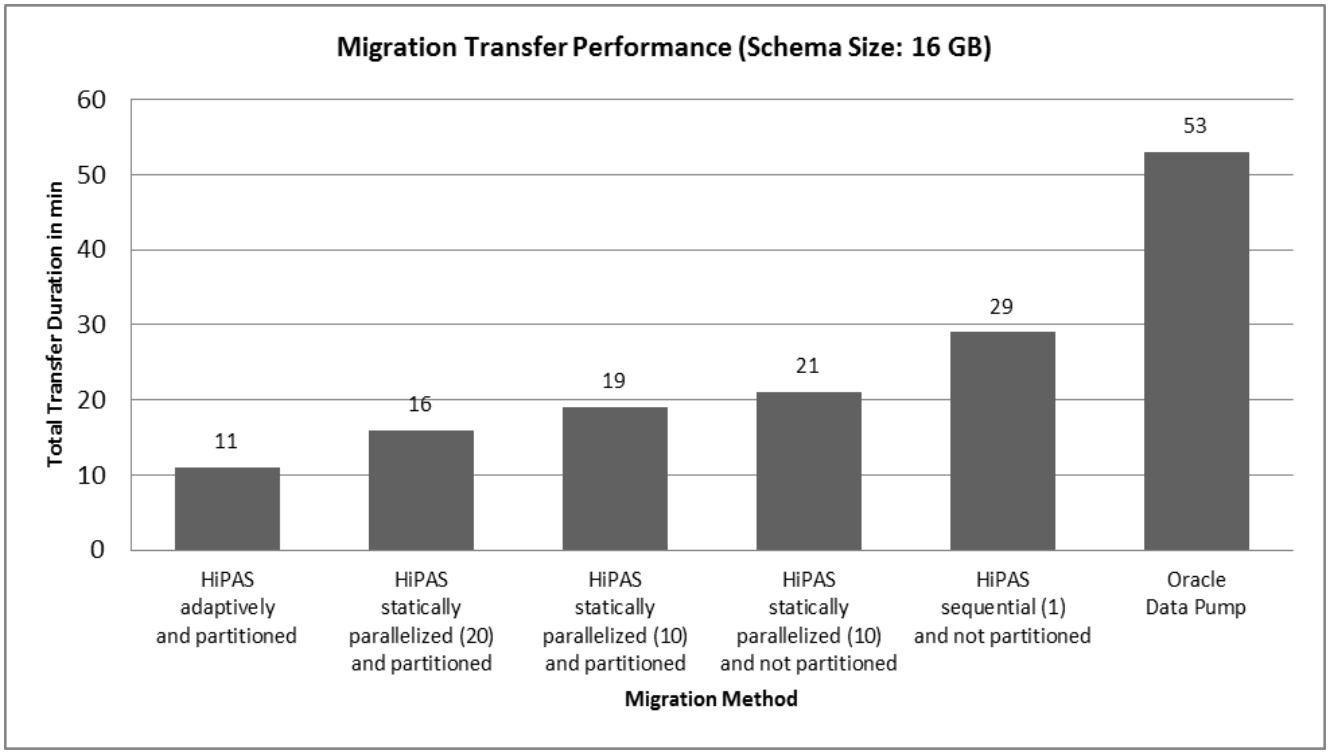


Figure 14. Transfer Performance for a 16GB Schema (B.1).

noticeably reduces the transfer duration, which is indicative for our assumption of utilizing the available resources more efficiently by parallelizing. Comparing test run 3 and 4 shows that partitioning large tables for the transfer barely improves the overall performance, since the partitioning feature was implemented to improve the flexibility of HiPAS when its optimizer needs to adapt quickly to changing resource availabilities. Thus, the adaptive migration run with enabled partitioning of large tables performed best in terms of downtime shortness.

3) Performance Test (Test B.2): In addition to the 16 GB schema, we performed the same test runs with a schema size of 32 GB to evaluate how the adaptive capabilities work for a longer period of transfer time. The static parallelized runs have been performed as well and showed results proportional to test B.1, so we excluded them from Figure 15.

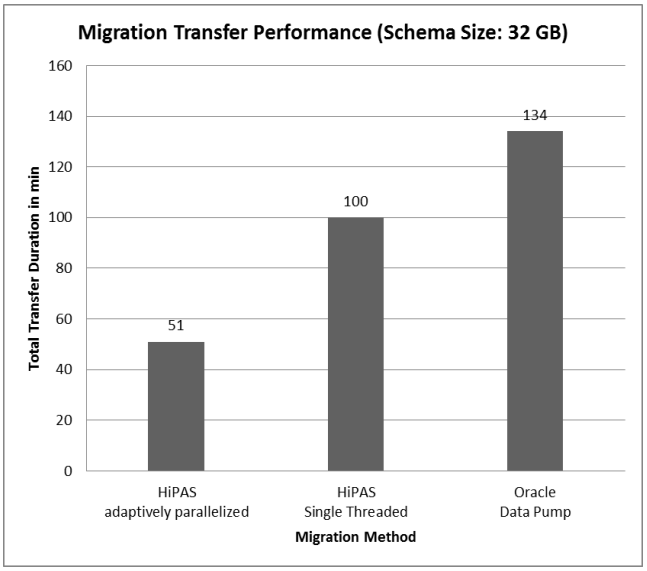


Figure 15. Transfer Performance for a 32GB Schema (B.2).

HiPAS, with enabled partitioning, adaptively transferred the schema within 51 minutes, compared to 2.23 hours needed by Data Pump, meaning this time HiPAS took 38% of Data Pump's transfer duration, whereas the single threaded configured HiPAS took about 75%. As a consequence, we assume that non-adaptive sequential and Data Pump migrations leave useful resources idle or need to be tuned manually. In addition to the introduced test runs for evaluation within the scope of this paper, we performed several further tests in customer environments achieving considerable results, especially for schemata storing large objects. In terms of network bandwidth, we reached transfer rates of 120 MB/s for each database link created on a 1 Gigabit Ethernet.

IX. Conclusion and Future Work

By designing and developing HiPAS, we applied adaptive software into the field of database migrations. We focused on self-optimization as the main adaptive property and implemented a system, which continuously optimizes the system utilization by adjusting the current amount of parallel workload. The observed environment is represented by monitoring information regarding the performance of the source and target database instances and their underlying storage systems. For optimizing the transfer phase, we state that implementing anticipatory capabilities into migration software using a MAPE-K feedback loop significantly improved the performance of migrations invoked on database layer, compared to a solely adaptive approach or non-adaptive migration software. Statically parallelized test runs did not adapt to changing utilization requirements, thus, performed less efficiently. The decisions taken in the design space for adaptive software and the paradigm of saving all migration metadata inside the database allows a highly reliable and transparent architecture, which supports an efficient interaction of all HiPAS migration components and the actual migration data. On the contrary, the implementation as a stored object leads to the disadvantage of having to develop separate implementations for different database systems. Therefore, we plan to build and evaluate further versions of HiPAS supporting different types of source and target systems. Another new version, we are working on, is intended to support online migrations, where the adaptive optimizer can be leveraged to utilize the source system up to a degree, which does not affect its availability and response time during productive use. Our results serve as a contribution for all practitioners in the field of database migrations, as well as researchers on self-adaptive software and their various fields of application.

Acknowledgment

We strongly like to thank all members of the Pasolfora Performance Research and Innovation Group (PPRG) for the support and possibility of performing the countless number of test and demo migrations during the development and evaluation of HiPAS. Furthermore, we thank Prof. Dr. Michael Höding of the Brandenburg University of Applied Sciences for giving scientific relevant input when mapping adaptive insights to the requirements of offline database migrations.

References

[1] H. Müller, A. Prusch, and S. Agel, "HiPAS: High Performance Adaptive Schema Migration – Evaluation of a Self-Optimizing Database Migration," in *DEPEND 2014, The Seventh International Conference on Dependability*, 2014, pp. 41–50.

[2] H. M. Sneed, E. Wof, and H. Heilmann, *Softwaremigration in Praxis*. dpunkt.verlag, 2010.

[3] M. L. Brodie and M. Stonebraker, *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., 1995.

[4] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE software*, vol. 16, no. 5, pp. 103–111, 1999.

[5] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software*

Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 1–32.

[6] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[7] M. Eastwood, J. Scaramella, K. Stolarski, and S. M., "Worldwide Server Market Revenues Decline -6.2% in Second Quarter as Market Demand Remains Weak, According to IDC." International Data Corporation, 2013 [Online]. Available: http://www.reuters.com/article/2013/08/28/ma-idc-idUSnBw276497a+100+BSW20130828 [Accessed: 29-May-2015]

[8] J. D. Little, "A proof for the queuing formula: L= λ W," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[9] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, vol. 84. Prentice-Hall Englewood Cliffs, 1984.

[10] R. et al. Alves, *Information Storage and Management - Storing, Managing, and Protecting Digital Information*. EMC Education Services, 2009.

[11] R. M. Karp, *Reducibility Among Combinatorial Problems*. Springer, 1972.

[12] "Fraunhofer. Adaptive Systems. Fraunhofer Institute for Embedded Systems and Communication Technologies." [Online]. Available: http://www.esk.fraunhofer.de/de/kompetenzen/adaptive_systeme.html [Accessed: 29-May-2015]

[13] J. A. Martın H, J. de Lope, and D. Maravall, "Adaptation, anticipation and rationality in natural and artificial systems: computational paradigms mimicking nature," *Natural Computing*, vol. 8, no. 4, pp. 757–775, 2009.

[14] N. Wiener, E. Henze, and E. H. Serr, *Kybernetik*. Econ-Verlag Düsseldorf, 1963.

[15] R. Rosen, *Anticipatory Systems*. Springer, 2012.

[16] M. Salehie and L. Tahvildari, "Self-adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, 2009.

[17] R. Laddaga, "Self-adaptive Software," Defense Advanced Research Projects Agency, 1997.

[18] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent systems*, vol. 14, no. 3, pp. 54–62, 1999.

[19] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology." IBM, 2001 [Online]. Available: http://people.scs.carleton.ca/ soma/biosec/readings/autonomic_computing.pdf [Accessed: 29-May-2015]

[20] "The 8 Elements." IBM [Online]. Available: http://www.personal.psu.edu/users/a/l/alw/autonomic/autonomic8.pdf [Accessed: 29-May-2015]

[21] "An Architectural Blueprint for Autonomic Computing," *IBM White Paper*. Citeseer, 2005 [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf [Accessed: 29-May-2015]

[22] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit, "A Design Space for Self-Adaptive Systems," in *Software Engineering for Self-adaptive Systems II*, Springer, 2013, pp. 33–50.

[23] M. G. Hinchey and R. Sterritt, "Self-Managing Software," *Computer*, vol. 39, no. 2, pp. 107–109, 2006.

[24] M. Parashar and S. Hariri, "Autonomic Computing: An overview," in *Unconventional Programming Paradigms*, Springer, 2005, pp. 257–269.

[25] "Enqueue: HW, Segment High Water Mark - Contention." Oracle, 2009 [Online]. Available: http://docs.oracle.com/cd/B16240_01/doc/doc.102/e16282/oracle_database_help/oracle_database_wait_bottlenecks_enqueue_hw_pct.html [Accessed: 29-May-2015]

[26] G. Claborn, W. Fisher, C. Palmer, J. Stenoish, and R. Swonger, "Data Pump in Oracle Database 11g Release 2: Foun-dation for Ultra High-Speed Data Movement Utilities." Oracle, 2010 [Online]. Available: http://download.oracle.com/otndocs/products/database/enterprise_edition/utilities/pdf/datapump11gr2_techover_1009.pdf [Accessed: 29-May-2015]

[27] K. Rich, "Oracle Database Utilities 11g Release 2." Oracle, 2014 [Online]. Available: http://docs.oracle.com/cloud/latest/db112/SUTIL.pdf [Accessed: 29-May-2015]

A Study on the Difficulty of Accounting for Data Processing in Functional Size Measures

Luigi Lavazza Sandro Morasca Davide Tosi
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
{luigi.lavazza, sandro.morasca, davide.tosi}@uninsubria.it

***Abstract*—The most popular Functional Size Measurement methods adopt a concept of "functionality" that is based mainly on the data involved in functions and data movements. Functional size measures are often used as a basis for estimating the effort required for software development. However, Functional Size Measurement does not take directly into consideration the amount of data processing involved in a process, even though it is well-known that development effort does depend on the amount of data processing code to be written. Thus, it is interesting to investigate to what extent the most popular functional size measures represent the data processing features of requirements and, consequently, the amount of data processing code to be written. To this end, we consider three applications that provide similar functionality, but require different amounts of data processing. These applications are then measured via a few Functional Size Measurement methods and traditional size measures (such as Lines of Code). A comparison of the obtained measures shows that differences among the applications are best represented by differences in Lines of Code. It is likely that the actual size of an application that requires substantial amounts of data processing is not fully represented by functional size measures. In summary, the paper shows that not taking into account data processing dramatically limits the expressiveness of the functional size measures. Practitioners that use size measures for effort estimation should complement functional size measures with measures that quantify data processing, to obtain precise effort estimates.**

Keywords- functional size measurement; Function Point Analysis; IFPUG Function Points;COSMIC method.

I. INTRODUCTION

Functional Size Measurement (FSM) methods aim at quantifying the "functional size" of an application. Such size should represent the "amount of functionality" provided to the user by a software application. It is quite reasonable to expect that the "amount of functionality" is to some extent correlated to the amount of data processing performed by the application. In this respect, there are some doubts that FSM methods properly account for the amount of data processing when sizing software applications [1].

In fact, the most popular FSM methods adopt a concept of "functionality" that is based mainly on the number of operations that can be performed by the users via the software application and the amount of data managed by the application. More precisely, the most popular FSM methods take into account

- the processes, named Elementary Processes (EP) in IFPUG and Functional Processes (FPr) in COSMIC;
- the data that cross the boundary of the application being measured or that are used (i.e., read or written) in the context of a process.

In this paper, we consider the most widely known and used FSM methods:

- IFPUG (International Function point User Group) Function Points [2][3], which were originally proposed in 1979 [4] and are widely known and used today;
- Mark II Function Points [5][6], which were proposed to improve Function Points;
- COSMIC (Common Software Measurement International Consortium) [7], which aims at further improving the characteristics of functional size measures;
- Use Case Points [8], a method that was proposed for usage with the Objectory process [9] (which was then incorporated into UML).

Quite noticeably, none of the mentioned methods satisfactorily considers the amount of data processing involved in a process. As a matter of fact, some methods propose an adjustment of the size based on the characteristics of data processing, but quite imprecisely and ineffectively, as discussed in Section VIII, while other methods do not take the amount of data processing into account at all.

The goal of the paper is to provide evidence, based on examples, that not considering data processing dramatically limits the expressiveness of functional size measures.

The core of the paper can be described as follows:

- Three applications are specified. These applications are similar with respect to the aims and functionality offered to the user, but they are very different in the amount of data processing required.
- The considered applications are modeled and measured according to four different functional size measurement methods.
- It is highlighted that the applications have the same functional size measures, even though the amount of functionality to be coded is dramatically different.
- When measured via Lines of Code, it is apparent that the implementations of the applications have quite different

sizes. The reason is that –quite obviously– more data processing requires more code.

It is unlikely that the additional code required for additional data processing requires a negligible additional amount of development effort. Thus, using only the functional size to estimate development effort for applications that require a substantial amount of data processing may lead to large and dangerous effort underestimations.

Currently, development effort is commonly estimated based on the functional size and possibly some other environmental factors, but without taking in due consideration the amount of data processing required. Sometimes this practice is justified by the fact that the application to be developed is estimated using productivity models derived from the analysis of previous projects in the same application domain. There is an underlying assumption that applications in the same domain require approximately the same amount of data processing. In this paper, we show that the contrary is true, by measuring programs that belong to the same domain.

The difficulty to quantitatively represent the amount of data processing appears as an intrinsic –though not generally recognized– limit of FSM methods. It should be noted that this paper does not aim at proposing a method to account for data processing in functional size measures. Instead, we aim at providing some evidence of the problem, to raise the awareness of the limits of FSM methods and solicit research efforts towards working out solutions. At the same time, we warn practitioners about the risks connected with assuming that the amount of data processing is somewhat automatically incorporated in traditional functional size measures, as such assumption could lead to severely underestimating the actual size of the application to be developed.

The paper is structured as follows. Section II reports a few basic concepts of functional size measurement. Section III illustrates the case studies used in the paper. Section IV describes the models and measures of the considered applications: the collected measures are then compared in Section V. In Section VI, additional examples showing the limitations of FSM methods in accounting for data processing are given. Section VII discusses the alternatives that should be considered for complementing standards functional size measures with measures that represent data processing. Section VIII accounts for related work. Finally, Section IX draws conclusions and briefly sketches future work.

This paper is an extended version of a previous paper [1]. Here, we use two additional sizing methods (namely Use Case Points and Mark II Function Points): this allows us to generalize the presented results. Moreover, we considered an additional application in the board games with artificial intelligence domain, which confirms the results given in [1], thus increasing the reliability of our conclusions. To this end, a discussion of different domains has also been added in Section VI.

II. FSM CONCEPTS

Functional Size Measurement methods aim at providing a measure of the size of the functional specifications of a given software application. Here, we do not need to explain in detail the principles upon which FSM methods are based. Instead, for our purposes it is important to consider *what* is actually measured, i.e., the model of software functional specifications that is used by FSM methods.

A. Function Point Software Model

The model used by Function Point Analysis (FPA) is given in Figure 1. Briefly, Logical files are the data processed by the application, and transactions are the operations available to users. The size measure in Function Points is computed as a weighted sum of the number of Logical files and Transactions. The weight of logical data files is computed based on the Record Elements Types (RET: subgroups of data belonging to a data file) and Data Element Types (DET: the elementary pieces of data). The weight of transactions is computed based on the Logical files involved –see the FTR (File Type Referenced) association in Figure 1– and the Data Element Types used for I/O.

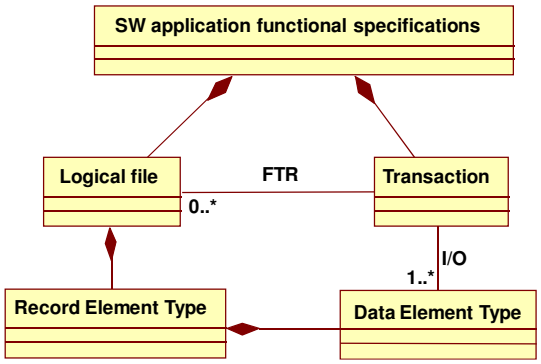


Figure 1. The model of software used in Function Point Analysis.

It is possible to see that in the FPA model of software, data processing is not represented at all.

B. COSMIC Software Model

The model used by COSMIC is given in Figure 2.

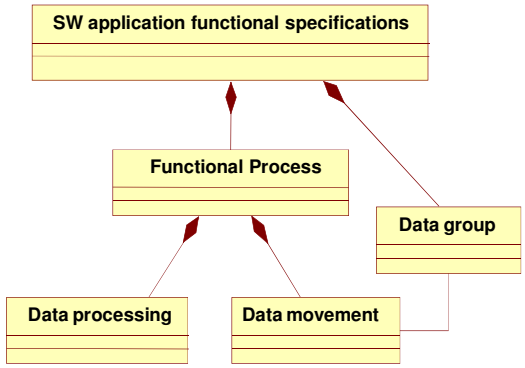


Figure 2. The model of software used by the COSMIC method.

The size of the functional specification expressed in COSMIC function points (CFP) is the sum of the sizes of functional processes; the size of each functional process is the number of distinct data movements it involves. A data movement concerns exactly one data group.

Although represented in Figure 2, neither data groups nor data processing are directly used in the determination of an application's functional size. In particular, data processing is not measured, since the COSMIC method assumes that a fixed amount of data processing is associated with every data movement; however, this is not the case in the examples considered in this paper.

C. Mark II FP Model

Symons proposed Mark II Function Points as an improvement of Albrecht's FPA in 1988 [5].

The application to be measured is modeled (see Figure 3) as a set of "logical transactions," which are essentially equivalent to IFPUG FP transactions and COSMIC functional processes. Each logical transaction is characterized in terms of the number of input DET, the number of output DET, and the number of Data Entity Types Referenced. In the Mark II FP model, DET have the same meaning as in the IFPUG FP model, while entities replace logical files (however, today the meaning associated with logical files is the same as that of Symons's entities).

The functional size in Mark II FP is the weighted sum, over all Logical Transactions, of the Input Data Element Types (N_i), the Data Entity Types Referenced (N_e), and the Output Data Element Types (N_o).

So the Mark II FP size for an application is:

$$Size = W_i \times \Sigma N_i + W_e \times \Sigma N_e + W_o \times \Sigma N_o$$

where 'Σ' means the sum over all Logical Transactions, and the industry average weights are $W_i = 0.58$, $W_e = 1.66$, and $W_o = 0.26$ [6].

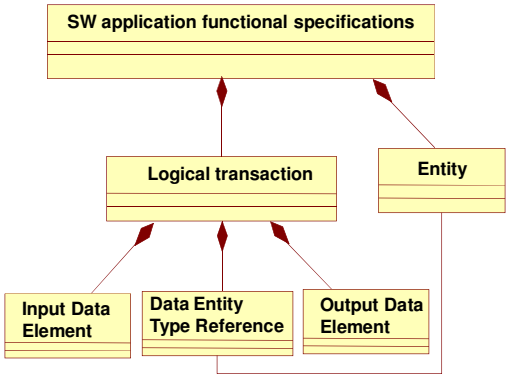


Figure 3. The model of software used in Mark II FP measurement.

D. Use Case Points Software Model

Use Case Points (UCP) were proposed by Karner to measure the size of applications specified via use cases [8]. Thus, the model of software considered for UCP measurement is centered on the concept of use case [9], as shown in Figure 4.

The UCP measurement process involves two phases. In the first one, the given application is measured in Unadjusted Use Case Points (UUCP). In the second one, the size expressed in UUCP can be "corrected" with the Technical Complexity Factor (TCF), which represents how difficult to construct the program is, and the Environmental Factor (EF), which represents how efficient our project is.

To compute size in UUCP, the considered factors are the application's users, the use cases, the transactions carried out in each use case, and the "analysis objects" (i.e., (interface, control, and entity objects, as defined in Objectory process [9]) used to realize the use case.

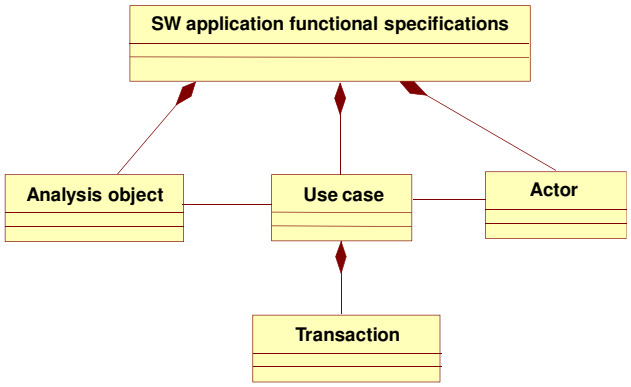


Figure 4. The model of software used in Use Case Points measurement.

The functional size in UUCP is the weighted sum of the Actors and the Use Cases. Both use cases and actors are weighted according to their "complexity." The complexity of actors is determined by nature of the actor (human or external system) and the type of interaction (e.g., via a GUI, or a command line interface). The complexity of use cases is determined by the number of involved transactions and the number of analysis objects needed to implement the use case.

The size in Use Case Points is calculated as follows.

$$TCF = 0.6 + 0.01 \times \Sigma FC_i \times W_i$$

$$EF = 1.4 - 0.03 \times \Sigma FE_i \times W_i$$

$$UCP = UUCP \times TCF \times EF$$

Where FC_i are 13 factors contributing to complexity and FE_i are 8 factors contributing to efficiency. W_i are the weights (integer value in the [0,5] interval) assigned to factors.

The details of the measurement can be found in [8].

III. Case Studies

In this section, we describe the functional specifications of the software applications that will be used to test the functional sizing ability of FPA and COSMIC.

The chosen applications are programs for playing board games against the computer. They are similar as for the functionality they provide, but they require different amounts of data processing.

The specifications that apply to both applications are as follows.

- The program lets a human player play against the computer.
- The program features a graphical interface in which the game board is represented.
- The player makes his/her moves by clicking on the board. Illegal moves are detected and have no effect. As soon as the human player has made a move, the computer determines its move and shows it on the board.
- When the game ends, the result is shown, and the player is asked if he/she wants to play another game.

The use case diagram of the considered applications is shown in Figure 5. From the point of view of the player, two main operations are available: to initiate a new game and to perform a move. In the latter case, the program will also compute its move. In both cases, the board is updated and displayed. A minor functionality of the program allows the player to show a few pieces of information concerning the application and its authors.

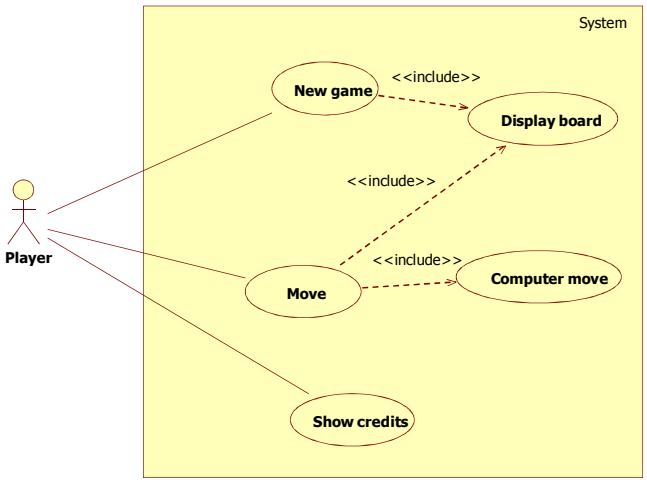


Figure 5. Use case diagram of the considered applications.

It is worth stressing that the use case diagram in Figure 5 describes all the considered applications, which differ from each other only for the implemented game, hence for the logic employed to compute the moves.

A. A Software Application to Play Tic-tac-toe

Tic-tac-toe is a very simple, universally known game. It is played on a 3×3 board, as shown in Figure 6.

Each player in turn puts his/her token in a free cell. The first player to place three tokens in a line (horizontally, vertically, or diagonally) wins. When the board is filled and no three-token line exists, the match is tie.

Playing Tic-tac-toe is very simple. In fact, to play optimally, a software program has just to evaluate the applicability of a short sequence of rules: the first applicable rule determines the move.

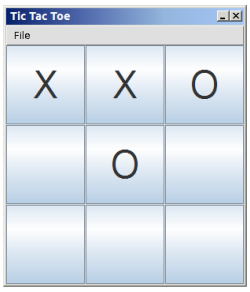


Figure 6. Tic Tac Toe playing board.

There are a few possible rule sequences: the one implemented in the considered application is the following:

1) If there is a line (row, column, or diagonal) such that two cells contain your token and the third cell X is empty, put your token in the free cell X, to win.
2) If there is a line in which your opponent has two tokens and the third cell X is free, put your token in the free cell X, so to prevent your opponent from winning at next turn.
3) If there is a move that lets you gain a winning position, make it.
4) If there is any move such that the adversary will not be able to gain a winning position at next turn, make such move. If possible, put the token in central cell.
5) If there is any cell free, put your token there.

A position is a winning one for a player when there are two lines each occupied by two tokens of the player, while the third cell is free.

The code that implements the playing logic described above is very simple and very small: we can expect that a few tens of lines of code are sufficient to code the game logic.

B. A Software Application to Play "five in a row"

Five in a row (aka Gomoku) can be seen as a generalization of Tic-tac-toe. In fact, it is played on a larger board (typically 19×19, as in Figure 7) and the aim of the game is to put five tokens of a player in a row (horizontally, vertically, or diagonally).

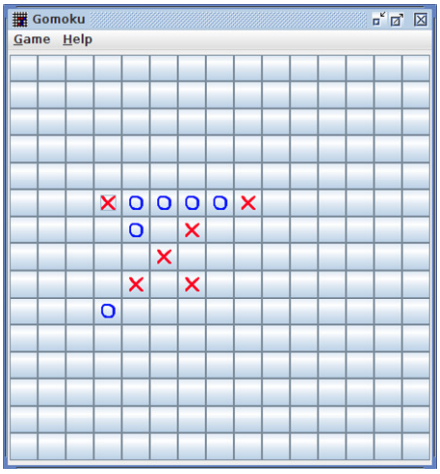


Figure 7. Gomoku playing board.

The functional specifications of Gomoku are exactly the same as the specifications of Tic-tac-toe, except that the size of the board is larger and the number of tokens to put in a row is 5 instead of 3.

The combinations of tokens and free cells that can occur on a Gomoku board are many more than in a Tic-tac-toe game. Accordingly, a winning strategy is much more complex, as it involves considering a bigger graph of possibilities.

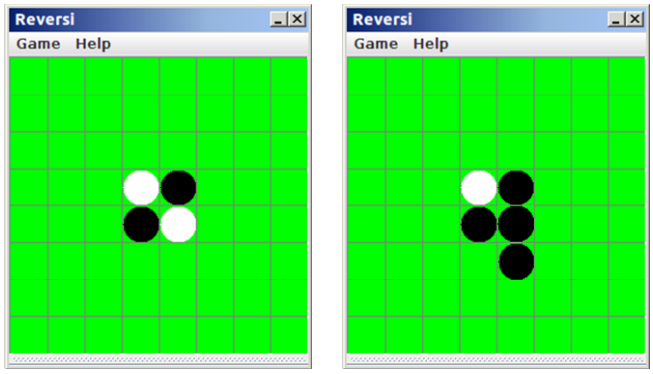
As a matter of fact, Gomoku has been a widely researched artificial intelligence research domain, and there are Gomoku professional players and tournaments.

Accordingly, we can safely state that Gomoku is a much more complex game than Tic-tac-toe and requires a large amount of processing, so that the machine can play at a level that is comparable with that of a human player.

On the contrary, Tic-tac-toe is a very simple game: you do not need to be particularly smart to master it and always play perfectly.

C. A Software Application to Play "Reversi"

Reversi (aka Othello) is played on an 8×8 board. The initial configuration is shown in Figure 8 a). Suppose player A has black tokens. At his/her turn, player A has to put its token in a position so as to form a horizontal, vertical, or diagonal line of adjacent tokens that has black tokens at the extremes and includes only white tokens (at least one). As an effect of the move, the white tokens between the black extremes become black. For instance, in the situation shown in Figure 8 a), player A could place his/her black token below the rightmost white token: such token is between two black tokens and becomes black, as shown in Figure 8 b). The game is named "Reversi" because usually the tokens are black on one side and white on the opposite one, so to change the color of a token you reverse it.



a) b)

Figure 8. Reversi playing board.

The strategy required to win a Reversi game is definitely more complex than the strategy required to play Tic-tac-toe. However, it is simpler than the strategy required for playing Gomoku, as the move search space is smaller.

IV. Application Sizing

A. Measurement of the Tic-tac-toe Application

Let us apply the FSM methods described in Section II to measure the Tic-tac-toe specifications given in Section III.A above.

1) Measuring Tic-tac-toe with IFPUG Function Points.

The software model to be used includes just a Logical data file: the board, which is a matrix of cells, each having one of three possible values (circle, cross, free). So, it is easy to see that there is only one Logical data file (the board), which is a simple Internal Logical File (ILF), contributing 7 FP.

The software model to be used involves the following elementary processes:

- Start a new game
- Make a move
- Show credits.

Start a new game is a simple External Input (EI), contributing 3 FP. Make a move is a simple external output, contributing 4 FP. One could wonder whether this operation should be considered an input (because the move involves inputting a position) or an output (because of the computation and visualization of the move by the computer). We consider that the latter is the main purpose of this transaction, which is thus an external output. Show credits is a simple External Query (EQ), contributing 3 FP.

In summary, the FPA size of the Tic-tac-toe application is 17 FP.

2) Measuring Tic-tac-toe with the COSMIC Method

The COSMIC functional processes of the application are the same as the FPA elementary processes. When measuring the application using the COSMIC method, we have to consider the data movements associated with each functional process:

- Start a new game involves clearing the board and possibly updating it, if the computer is the first to move (a Write) and showing it (a Read and an Exit). Therefore, this functional process contributes 3 CFP.
- Make a move involves entering a move (an Entry), updating the board with the human player move (a Write), reading it (a Read), and then updating it again with the computer move and showing it (an Exit). In addition, if a move concludes the game, the result is shown (an Exit). Therefore, this functional process contributes 5 CFP.
- Show credits involves the request to show credits (an Entry), reading the credits (a Read) and outputting them (an Exit). Thus, this functional process contributes 3 CFP.

In summary, the COSMIC size of the Tic-tac-toe application is 11 CFP.

3) Measuring Tic-tac-toe with Use Case Points

The Tic-tac-toe application has one user, who interacts with the system through a graphical user interface. According to UCP rules, such user is a complex actor, with weight 3.

The Tic-tac-toe application has three use cases, as shown in Figure 5. All of these use cases have 3 or fewer transactions and can be realized with less than 5 analysis objects; hence they are simple and their weight is 5.

So, the size of Tic-tac-toe is 3+5+5+5=18 UUCP.

TCF and EF involve several factors. However, only the "Complex internal processing" factor of TCF is relevant for our study, so we assume that all the factors considered in the TCF and EF have value 3, i.e., average relevance. As a consequence TCF×EF is $0.99 + 0.01 \times CIP$, where CIP is the value of the Complex Internal Processing.

The Complex Internal Processing factor is supposed to represent the complexity of the processing that is carried out in the application. It is rated on a scale 0, 1, 2, 3, 4, and 5. Unfortunately, in the original definition, Karner did not provide criteria to rate Complex internal processing; therefore, different persons could rate the same application differently. Tic-tac-toe surely is a very simple application, but it is very difficult to say if its Complex Internal Processing factor should be rated 0 or 1. So, we can conclude that the size of Tic-tac-toe is $18 \times (0.99 + 0.01$ CIP), that is, either 17.82 or 18, depending on the value assigned to CIP.

4) Measuring Tic-tac-toe with Mark II Function Points

To size the Tic-tac-toe application using Mark II FP, it is first necessary to identify the involved entities and Logical transactions. This is very easy, since we have only two entities (the board and the credits) while the logical transactions correspond to IFPUG FP transactions, COSMIC functional processes and UCP Use Cases (i.e., New game, Move, Show credits).

The size is computed according to the number of input data, entities referenced and output data as shown in TABLE I. While New game and Show credits have just one input (the event that triggers the operation), Move has two inputs: the row and column where the player puts his/her token.

New game and Show credits also have just one output (the board and the credits' text, respectively); Move outputs the board and the users' tokens, or a diagnostic message (when the player clicks on an already occupied cell).

New game and Move access the board entity, Show credits accesses the credits entity.

TABLE I. MEASURES OF TIC-TAC-TOE APPLICATION IN MARK II FP

Logical transaction	***Ni***	***Ne***	***No***	***MKII FP***
New game	1	1	1	2.5
Move	2	1	3	3.6
Show credits	1	1	1	2.5
Total				**8.60**

In conclusion, the application to play Tic-tac-toe has size 8.60 MKII FP.

5) Tic-tac-toe Code Measures

Since we are also interested in indications concerning the amount of computation performed by the application, we selected an open source implementation of Tic-tac-toe and measured it.

To evaluate the "physical" size of the Tic-tac-toe application, we looked for an open source application that implements the specifications described above. Two such applications are the programs available from [10] and [11]. To make the considered program functionally equivalent to the other applications, we performed a merge of the code from [10] and [11]. The main measures that characterize the obtained code are given in TABLE II.

TABLE II. MEASURES OF THE TIC-TAC-TOE APPLICATION CODE

Measures	**Tic-tac-toe**	
	Total	***AI part***
LoC	286	146
Number of Java statements	187	101
McCabe (method mean)	3.6	4.5
Num. classes	2	1
Num. methods	26	13

In TABLE I (and in TABLE II), column "AI part" indicates the measures concerning exclusively the part of the code that contains the determination of the computer move.

We reported both the number of lines of code and the number of actual Java statements: the latter is a more precise indication of the amount of source code, since it does not consider blank lines, comments and lines containing only syntactic elements, like parentheses. We also reported the mean value of McCabe complexity of methods.

B. Measurement of Gomoku Application

Let us measure the Gomoku specifications given in Section III.B above

1) Measuring Gomoku with IFPUG Function Points and COSMIC

The functional size measures of the Gomoku application are exactly the same as the measures of the Tic-tac-toe application. In fact, the specifications of the two applications are equal, except for the board size and winning row size, which do not affect the measurement, because both IFPUG FPA and COSMIC consider data types, not the value or number of instances.

2) Measuring Gomoku with Use Case Points

Gomoku has the same actor and use cases as Tic-tac-toe. Therefore, the size of Gomoku measured in UUCP is equal to Tic-tac-toe's.

As for Tic-tac-toe, we assume that the factors that determine TCF and EF are all average, except for the CIP; therefore, TCF×EF is $0.99 + 0.01 \times CIP$.

Gomoku is definitely a much more complex game than Tic-tac-toe; therefore, the Gomoku playing program has to perform quite complex processing to achieve an acceptable playing level. We can therefore assign the Complex Internal Processing a high rating, even though it is not clear whether we should set CIP=5 or CIP=4. In conclusion, the size of Gomoku is 18.54 or 18.72, depending on the value of CIP.

3) Measuring Gomoku with Mark II FP

The Gomoku application is characterized by the same actors, use cases, transactions and entities as the Tic-tac-toe application. Therefore, they have the same size measure expressed in Mark II FP.

4) Gomoku Code Measures

As for Tic-tac-toe, we selected an open source implementation of Gomoku and measured it. More precisely, we looked for a program capable of sophisticated "reasoning" that lets the program play at the level of a fairly good human player. One such application is the Gomoku Java program available from [12].

The main measures that characterize the code are given in TABLE III.

TABLE III. MEASURES OF THE GOMOKU APPLICATION CODE

Measures	Gomoku [12]	
	Total	*AI part*
LoC	859	373
Number of Java statements	419	212
McCabe (method mean)	2.6	5.4
Num. classes	17	3
Num. methods	83	25

Measures in TABLE III were derived using the same tools and have the same meaning as the measures in TABLE II.

C. Measurement of Reversi Application

1) Measuring Reversi with IFPUG Function Points and COSMIC

The functional size measures of the Reversi application are exactly the same as the measures of the Tic-tac-toe and Gomoku applications. In fact, the specifications of the three applications are characterized by the same basic functional components.

2) Measuring Reversi with Use Case Points

Reversi has the same actor and use cases as Tic-tac-toe and Gomoku. Therefore, the size of Reversi measured in UUCP is equal to Tic-tac-toe's and Gomoku's.

As for the other applications, we assume that the factors that determine TCF and EF are all average, except for the CIP; therefore, TCF×EF is $0.99 + 0.01 \times$ CIP.

Reversi is definitely more complex than Tic-tac-toe, but less complex than Gomoku. We can therefore assign the Complex Internal Processing a high rating, but not as high as Gomoku's. So, it is probably reasonable to set CIP=4 or CIP=3. In conclusion, the size of Reversi is 18.36 or 18.54, depending on the value of CIP.

Note that the value assigned to CIP is largely subjective: this is due to the fact that the definition of UCP does not provide precise guidelines for determining the values of TCF and EF factors.

3) Measuring Reversi with Mark II Function Points

When measuring the Reversi applications with Mark II FP, the same considerations reported for Tic-tac-toe and Gomoku apply. The only difference is that when the New game logical transaction is performed, the initial situation of the board is not empty, therefore we have 2 additional output DET associated to New game. This is shown in TABLE IV.

TABLE IV. MEASURES OF THE REVERSI APPLICATION IN MARK II FP

Logical transaction	Ni	Ne	No	MKII FP
New game	1	1	3	3.02
Move	2	1	3	3.6
Show credits	1	1	1	2.5
Total				**9.12**

In conclusion, the application to play Reversi has size 9.12 MKII FP.

4) Reversi Code Measures

Like with the other applications, we selected an open source Java implementation of Reversi [13] and measured it.

More precisely, the implementation of Reversi that we found [13] was richer than the implementations of Tic-tac-toe and Gomoku in functionality (e.g., it features a help function, the possibility of choosing the playing level and the dashboard color, etc.). To make the Reversi application comparable to the others, we simplified the implementation, deleting all the additional functions and the corresponding code.

The main measures that characterize the resulting code are given in TABLE V.

TABLE V. MEASURES OF THE REVERSI APPLICATION CODE

Measures	Reversi [13]	
	Total	*AI part*
LoC	419	218
Number of Java statements	290	180
McCabe (method mean)	3.1	4.4
Num. classes	6	4
Num. methods	36	17

Measures in TABLE V were derived using the same tools and have the same meaning as the measures in TABLE II and TABLE III.

V. COMPARISON OF MEASURES

The measures reported in Section IV and summarized in TABLE VI show that a few applications may have the same functional size, but very different code size: for instance, the Gomoku application is twice as big as the Tic-tac-toe application. Considering the nature of these applications, the difference in code is largely explained by the different amount of processing required. In the case of Tic-tac-toe, the number of possible moves is very small, as is the number of different possible configurations that can be achieved by means of a move: hence, every move computation has to explore a very small space. The contrary is true for the

Gomoku application. The consequence is that Gomoku requires an amount of code devoted to move computation that is more than twice as much as the code required by Tic-tac-toe. Reversi requires more data processing than Tic-tac-toe and less processing than Gomoku; accordingly, its implementation is bigger than Tic-tac-toe's and smaller than Gomoku's.

The collected measures are summarized in TABLE VI. Both measures concerning the complete application (column Total) and measures concerning the artificial intelligence part of the application (column AI) are given. It should be noted that the functional size makes sense only concerning the complete application, since it is not allowed in FSM methods to measure only a portion of the application (this would actually be possible with the COSMIC method, but the resulting measure would not be comparable to those of the complete applications, though).

TABLE VI. SUMMARY OF THE APPLICATIONS' MEASURES.

	Tic-tac-toe		**Reversi**		**Gomoku**	
	Total	***AI***	***Total***	***AI***	***Total***	***AI***
Data proc.	Average	Very low	Average	Medium -high	Average	High
Java statem.	187	101	290	180	419	212
McCabe	3.6	4.5	3.1	4.4	2.6	5.4
Classes	2	1	6	4	17	3
Methods	26	13	36	17	83	25
IFPUG FP	17		17		17	
CFP	11		11		11	
UUCP	18		18		18	
UCP	17.8–18		18.4–18.5		18.5–18.7	
Mark II FP	8.60		9.12		8.60	

These observations suggest a few important considerations, which are reported below.

A. *Functional Size and Data Processing*

The definitions of IFPUG FP, COSMIC FP, Mark II FP, and UUCP do not properly take into account the amount of processing required by software functional specifications. If we plot the three considered applications in a Cartesian plane, where the axes represent the amount of required data processing and the functional size (expressed via any functional size measure), we get the situation described in Figure 9 (note that the y axis is not in scale). It appears that there is no relationship that links the functional size and the amount of processing required.

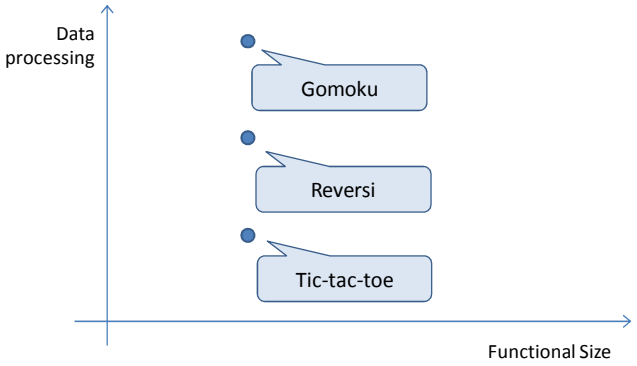


Figure 9. Plot of data processing vs. functional size for the considered applications.

For the sake of precision, we must note that Mark II FP and UUCP measures are not equal for all the applications, but are only slightly different, while the differences in terms of data processing are fairly large.

If we plot the three considered applications in a Cartesian plane, where the axes represent the amount of required data processing and the physical size (expressed in LoC, or number of statement, or in number of methods, etc.), we get the situation described in Figure 9 (note that axis scales are just indicative). It is possible to see that there is a clear relationship between the physical size and the amount of processing required.

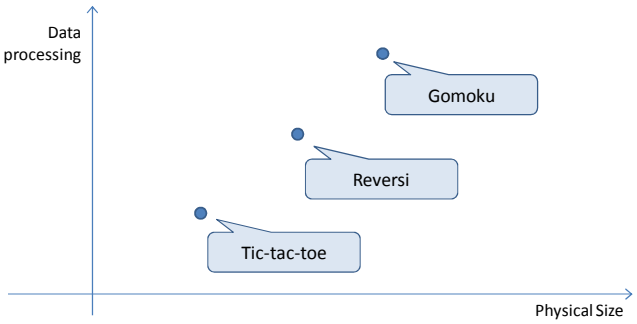


Figure 10. Plot of data processing vs. physical size for the considered applications.

If we assume –as is generally accepted– that the effort required to implement a software application is related to the number of Lines of Code to be written, the possibility of having widely different sizes in LoC for applications that have the same functional size implies that functional size is not a good enough predictor of development effort.

B. *Mark II FP Measures*

In the considered cases, Mark II FP size measures indicate that Reversi is marginally bigger than the other applications. This is misleading when considering that the Gomoku application is actually much bigger than Reversi. In addition, the difference with respect to Tic-tac-toe's size does not give a proper idea of the actual difference: the implementation of Reversi is 55% bigger than Tic-tac-toe's, while the functional size in Mark II FP is only 6% bigger.

C. Effort Required for Non-coding Activities

The observation reported in Section V.A above does not apply only to the coding phase. In fact, the difference in the number of classes and methods (shown in TABLE VI) suggests that also the effort required by design and testing activities is better estimated based on measures that represent the size of the code structure –like the number of classes– rather than the functional size.

D. The Explanation Power of TCF

In the analyzed cases, the correction to UUCP due to the TCF appears largely insufficient. In fact, assigning to CIP the biggest possible value (i.e., 5) for Gomoku and the smallest (i.e., 0) for Tic-tac-toe causes the size of Gomoku (18.72 UCP) to be only 5% bigger than the size of Tic-tac-toe (17.82 UCP). Such difference does not appear to be able to predict the difference in terms of Java statements to be written, as the AI code of Gomoku is twice as big as the AI code of Tic-tac-toe.

We should note that in this paper we considered Unadjusted Function Points, as defined in the ISO standard [3]. However, the IFPUG also defines "adjusted" Function Points, which are obtained by applying to the unadjusted measure a value adjustment factor (VAF) that is based on a few characteristics of the application being measured, including "Complex Processing" [2]. Actually, the definition of UCP's TCF and EF were inspired by Function Points' VAF. The considerations reported above concerning the representativeness of TCF apply to VAF as well. The "Complex Processing" component of VAF affects the size by less than 8%: too little to explain the observed differences in the considered applications' code size.

E. The Explanation Power of McCabe Complexity

As a final remark, we can observe that mean McCabe complexity is fairly similar in all the considered applications. The mean McCabe complexity of the AI part of the applications increases with the amount of data processing required by the games, but the differences are very small: from 4.5 of Tic-tac-toe to 5.4 of Gomoku. This means that applications dealing with more complex games (like Gomoku) do not need code that is much more complex (in McCabe's sense), but just more code. In other words, it is the difference in the amount of data processing, not in the complexity of the processing that is relevant, and that existing functional size measures fail to represent.

VI. ADDITIONAL EVIDENCE

The problems described above are at the level of elementary processes (alias transactions, alias functional processes). Namely, the problem with the considered board games is located in the Move process, which has the same functional size in all applications, but requires quite different data processing in the three considered applications.

Readers might wonder whether the described problem is due to the nature of the considered applications, which involve the usage of artificial intelligence. Actually, the same type of problem can be found in different application domains. Let us consider the measurement of source code. Several processes that are frequently found in measurement programs share the same set of properties, namely:

- Inputs: the request to measure and the name of the source code file to be measured.
- Output: the value of the measure.
- Data read: the code file.

Examples of such processes are the measurement of LoC, the measurement of Non commenting LoC (i.e., LoC not including comments), the measurement of McCabe complexity and the measurement of the coupling between objects (CBO) [14].

It is easy to see that these processes have the same functional size, whatever measure they compute. More precisely, they all have the same functional size if IFPUG FP, COMSIC FP or Mark II FP are used. If UCP are used, the sizes could differ of up to 5%, because of differences in the "Complex Internal Processing" factor.

However, different measures require different amounts of data processing:

- Total LoC: the processing is extremely simple, as it just involves counting the number of 'new line' characters.
- Non commenting: the required processing is more complex than in the former case, but still rather simple. In fact it is sufficient to recognize the beginning and end of comments and exclude lines that are entirely included in comments.
- McCabe complexity: the processing is more complex than in the previous case, since syntax analysis is required to recognize functions (or procedures or methods, depending on the programming language) and decision statements (if, while, for, etc.). The computation of McCabe complexity is usually performed by first parsing the code to obtain an abstract syntax tree, and then visiting the tree to count the relevant syntactic elements (if, while, etc.).
- CBO: the processing is still more complex. In fact, semantic analysis of code is also required, in addition to syntax analysis. For instance, when a statement like `a = b.new_class(x,y,z);` is found in class C, it is necessary to understand the type (class) returned by method new_class, to properly count the number of dependencies of class C.

So, a program that measures McCabe complexity and CBO has the same functional size as a program that counts total LoC and non-comment LoC; however, it is quite clear that implementing the former program is much more demanding in terms of development effort, since a greater amount of data processing has to be implemented.

Similar examples in different domains are easily found. For instance, in the statistical domain, a few processes have to show a series of data concerning a time period, via different representations. All the processes have the same inputs, read similar data, and output similar information (although using different graphical styles): accordingly, all the processes have the same functional size, since graphical styles are irrelevant. However, some of these process could require a very small amount of data processing. For instance, the process that shows data via a bar chart (Figure 11)

consists of a simple loop: at every iteration a value is read and a bar of length proportional to the value is drawn. On the contrary, a process that shows the data via a "smooth" interpolation line, e.g., a lowess (locally weighted scatterplot smoothing) curve [15] (Figure 12) has to perform a definitely greater amount of data processing, since the computation of weighted linear least squares regression is required.

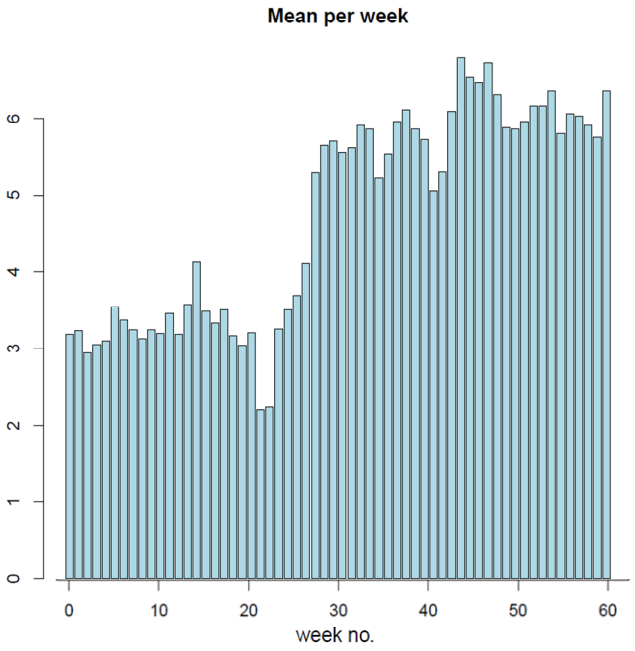


Figure 11. Output of a transaction that represents a time series via an histogram.

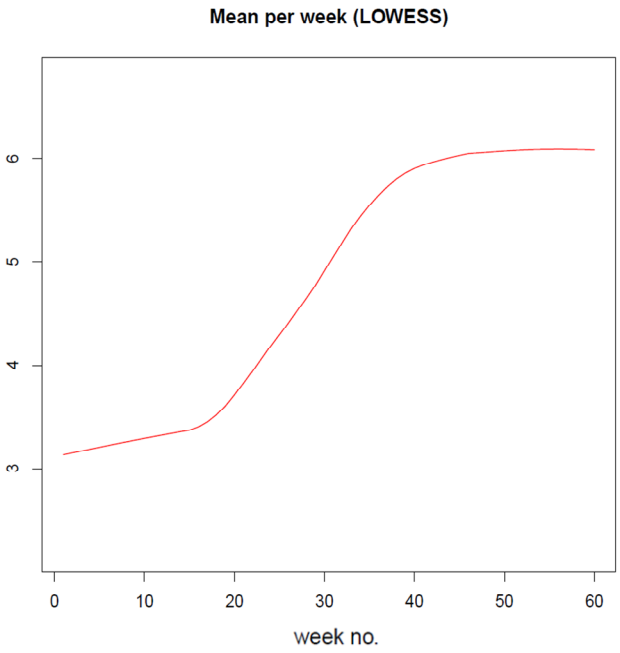


Figure 12. Output of a transaction that represents a time series via a LOWESS curve.

Summarizing, there are many examples of transactions (alias elementary processes, alias functional processes, etc.) whose functional size measure does not appear effective in representing the functionality delivered to the user, since the –quite variable– amount of data processing is not accounted for.

VII. Discussion: What Solutions Are Possible?

The usefulness of the evidence given in this paper stems from a few well-known facts:

- We need to estimate, during the early phases of a project, the overall software development effort.
- Development effort has been widely reported to be directly related to the size in LoC of software. Unfortunately, the size in LoC is not available in the early phases of projects, when estimates are most needed.
- Therefore, we need FSM methods, i.e., we need measures of functional specifications, because specifications are available in the early phases of projects.
- In this paper, we provided some evidence that current FSM methods appear limited in representing the amount of data processing required by functional specifications. Therefore, we need to somehow enhance FSM methods to remove such limitation.

So, we are facing the following research question: how can we improve FSM methods so that the delivered functional size measures account for the amount of data processing described or implied by the functional specifications?

This is an open research question, which calls for a substantial amount of further studies. In the following sections, we report a few observations, ideas, and evaluations that could be useful considering when tackling the problem.

A. *Software Models*

FSM methods –like any measurement method– are applied to models of the object to be measured. Hence, a rather straightforward consideration is that data processing must be represented in the model that describes the software application to be measured.

We can observe that the conceptual model of software proposed in the COSMIC method includes data processing, but no criteria or procedures for measuring data processing are given in the context of the COSMIC method.

In COSMIC, data processing is a sub-process of a functional process. Therefore, functional processes should be described in a manner that makes it possible to identify and measure the extent of data processing that occurs within a functional process.

Given the similarity of COSMIC functional processes and FPA elementary processes (or transactions) any technique used to enhance the expressivity of COSMIC models as far as data processing is concerned should be readily applicable to FPA models as well.

B. Software Specifications

A question that should be considered is if the information required for identifying and measuring data processing is always available from the software specifications that are derived from user requirements.

FSM methods use models of functional specifications: if functional specifications do not include information on data processing, neither do their models, and FSM methods will not be able to account for data processing.

So, another open question is the following: is it necessary to go beyond user requirements related specifications to be able to represent data processing? In other words: should elements of design be anticipated, to get better measures of the amount of data processing to be implemented?

C. Qualitative Knowledge

Current FSM methods are inherently quantitative. Even though some measurement activities –like deciding if two sets of data should be two RET of a unique logic file or they should belong to separate logic files– involve some subjectivity, they are always meant to provide measures (the number of ILF, RET, etc.) at the ratio level of measurement.

One could wonder if the use of more qualitative knowledge, derived through inherently subjective evaluations and expressed via ordinal scales, would be more suitable for expressing the relevant information concerning data processing.

For instance, after talking with stakeholders, an analyst could easily classify the functional process "Make a move" of the Tic-tac-toe application as very simple, while the same process of the Gomoku application could be classified as very complex.

D. Towards a Measure of Data Processing

As mentioned above, proposing a solution to the problem outlined above is very difficult. Here, we outline a few directions to be considered when addressing the problem.

A first consideration concerns the level of description of data processing. At a high level, the "variability" of the processes in terms of number of different cases to be considered could easily determine the amount of data processing required. Consider for instance a process that starts by identifying users: if the specifications indicate that the user can be identified in three different ways (e.g., by name, by social security number, and by email address) it is likely that it will have to process three times as much data as a process that identifies users in a single way.

Another observation concerns how to differentiate functionalities. A possibility is to account for the internal states a function has to deal with. In the case of tic-tac-toe, the number of states in which the game can be is quite small; on the contrary, the states of a Gomoku game are very numerous. Accordingly, the amount of computation could be proportional to the number of states, since the function has to properly deal with all states. However, the quantification of data processing could be further complicated by the presence of equivalent states, i.e., sets of states that are managed in the same way, so that having N or N+1 states in such sets would not affect the amount of processing required. For instance, a date increase function has to account for months having 28 (or 29), 30, or 31 days: the fact that there are 7 months having 31 days and just one having 28 days is irrelevant. In complex cases, identifying the relevant states could be very difficult; for instance, in Gomoku several token patterns can be identified, and each pattern calls for a specific strategy. So, the interesting states are the token patterns, but imagining in advance all the possible patterns is quite hard. A qualitative indication concerning the number of states would probably be more appropriate, in this case.

VIII. Related Work

Although several FSM methods (e.g., Mark II FP, NESMA and FiSMA) have been proposed as extensions or replacements of Function Point Analysis, very little attention has been given to the measurement of data processing.

A noticeable exception is the proposal of Feature Points by Capers Jones [16]. In this functional size measure, algorithms were added to the set of FPA basic functional components (ILF, EIF, EI, EO and EQ), and each component type was assigned a unique value, i.e., the notion of complexity was removed. The method was soon abandoned, mainly due to the difficulty of identifying algorithms, which are typically not documented in functional specifications.

Function Point Analysis and other methods –like Use Case Points [8]– introduce a mechanism for "adjusting" the size measure to take into account additional complexity factors that are likely to increase the effort required for implementation. In fact, among FPA value adjustment factors (VAF) we find "Complex Internal Processing," which represents to what degree the application includes extensive logical or mathematical processing. This mechanism is similar to what we need, but has a few shortcomings, including:

- In FPA the considered VAF's value increases the application size by 4% to 8%: at least one order of magnitude less than needed in the Tic-tac-toe vs. Gomoku case.
- The VAF applies to the whole application, so that it is not possible to distinguish simple and complex processes.

Noticeably, only the definition of unadjusted Function Points was standardized [3].

The Path measure [17][18] represents the complexity of processes in terms of the number of execution paths that are required for each process. Although this measure proved fairly effective in improving effort estimation based on functional size measures, it is not applicable in cases like those considered in this paper, since the alternative courses of the specified processes are not known.

IX. Conclusions

In this paper, we have shown by means of examples that FSM methods fail to represent the amount of data processing required by software functional specifications.

One could wonder how general are the results reported in the paper. As for this issue, we showed in Section VI that the

limits of FSM methods discussed in the paper apply to several application domains.

The work reported in the paper indicates that we need a measure that can complement traditional FSM methods to represent the amount of data processing that is needed to provide the required functionality.

We are interested in representing and quantifying the amount of data processing not because of an abstract interest in the definition of functional size measures, but because –as shown in the paper– data processing is logically related to code size, which in its turn is linked to the amount of development effort required to build a software application.

How to measure the amount of data processing required by the specifications of a software application is an open research question of great practical interest that should receive much more attention than it currently does.

ACKNOWLEDGMENT

The work presented here has been partly supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission and by project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software," funded by the Università degli Studi dell'Insubria.

REFERENCES

[1] L. Lavazza, S. Morasca, and D. Tosi, "On the Ability of Functional Size Measurement Methods to Size Complex Software Applications," 9th Int. Conf. on Software Engineering Advances - ICSEA 2014, October 12-16 2014, Nice. pp. 404-409.

[2] International Function Point Users Group. Function Point Counting Practices Manual - Release 4.3.1, January 2010.

[3] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, Geneva: ISO, 2003.

[4] A. J. Albrecht, "Measuring Application Development Productivity," Joint SHARE/ GUIDE/IBM Application Development Symposium, 1979, pp. 83-92.

[5] C. R. Symons, "Function point analysis: difficulties and improvements," IEEE Transactions on Software Engineering, 14.1, 1988.

[6] ISO/IEC 20968:2002, "Software engineering – Mk II Function Point Analysis – Counting Practices Manual," 2002.

[7] COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 4.0 Measurement Manual, April 2014.

[8] G. Karner, "Resource estimation for objectory projects," Objective Systems SF AB, 17. 1993.

[9] I. Jacobson, G. Booch, and J. Rumbaugh. "The Objectory Software Development Process," Addison Wesley, 1997.

[10] http://algojava.blogspot.it/2012/05/tic-tac-toe-game-swingjava.html, last accessed 15 May, 2105.

[11] http://sourceforge.net/projects/tictactoe-javab/files, last accessed 15 May, 2105.

[12] http://sourceforge.net/p/gomoku/, last accessed 15 May, 2105.

[13] https://reversi.java.net/, last accessed 15 May, 2105.

[14] S.R. Chidamber and C.F. Kemerer. "A metrics suite for object oriented design," IEEE Transactions on Software Engineering 20.6, 1994.

[15] W.S. Cleveland, "LOWESS: A program for smoothing scatterplots by robust locally weighted regression," American Statistician, 1981.

[16] C. Jones, "The SPR Feature Point Method," Software Productivity Research, 1986.

[17] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics," Innovations in Systems and Software Engineering, 4(1), 2008, pp. 31-43.

[18] L. Lavazza and G. Robiolo, "Introducing the evaluation of complexity in functional size measurement: a UML-based approach," ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement, September 2010.

www.iariajournals.org

International Journal On Advances in Intelligent Systems
issn: 1942-2679

International Journal On Advances in Internet Technology
issn: 1942-2652

International Journal On Advances in Life Sciences
issn: 1942-2660

International Journal On Advances in Networks and Services
issn: 1942-2644

International Journal On Advances in Security
issn: 1942-2636

International Journal On Advances in Software
issn: 1942-2628

International Journal On Advances in Systems and Measurements
issn: 1942-261x

International Journal On Advances in Telecommunications
issn: 1942-2601